NCS TIB 90-4

NATIONAL COMMUNICATIONS SYSTEM

TECHNICAL INFORMATION BULLETIN 90-4

Multiprotocol Gateway Optimization Report

MAY 1990

OFFICE OF THE MANAGER
NATIONAL COMMUNICATIONS SYSTEM

WASHINGTON, D.C. 20305

NCS TIB 90-4

NATIONAL COMMUNICATIONS SYSTEM

TECHNICAL INFORMATION BULLETIN 90-4

Multiprotocol Gateway Optimization Report

MAY 1990

OFFICE OF THE MANAGER
NATIONAL COMMUNICATIONS SYSTEM

WASHINGTON, D.C. 20305

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1990	Final

4. TITLE AND SUBTITLE

Multiprotocol Gateway Optimization Report

5. FUNDING NUMBERS

C-DCA100-87-C-0063

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Booz, Allen & Hamilton
4330 East West Highway
Bethesda, MD

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Communications System
Office of Technology & Standards
Washington, DC 20305-2010

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NCS TIB 90-4

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The OMNCS is working to provide standards within the data communication and telecommunication fields to increase the interoperability of diverse communication systems. These interoperability efforts improve the NS/EP communication capabilities of the nation by providing additional interconnectivity among federal telecommunication assets. One such effort is determining the feasibility of using the excess transmission capacity of the Integrated Serivces Digital Network (ISDN) out-of-band signaling channels to reconsitute disrupted networks. The purpose of this document is to present the optimized software component of the multiprotocol gateway, document the gateway software, and discuss potential enhancements to the multiprotocol gateway. This document also includes a user's guide for the gateway operator that describes the gateway menu display and available command options.

14. SUBJECT TERMS

Integrated Services Digital Network (ISDN) X.25
National Security & Emergency Preparedness (NS/EP)
Data Communications

15. NUMBER OF PAGES

232

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

NCS TECHNICAL INFORMATION BULLETIN 90-4

MULTIPROTOCOL GATEWAY OPTIMIZATION REPORT

MAY 1990

PROJECT OFFICER

GARY REKSTAD
Electronics Engineer
Office of NCS Technology
and Standards

APPROVED FOR PUBLICATION:

DENNIS BODSON
Assistant Manager
Office of NCS Technology
and Standards

FOREWORD

Among the responsibilities assigned to the Office of the Manager, National Communications System, is the management of the Federal Telecommunication Standards Program. Under this program, the NCS, with the assistance of the Federal Telecommunication Standards Committee identifies, develops, and coordinates proposed Federal Standards which either contribute to the interoperability of functionally similar Federal telecommunication systems or to the achievement of a compatible and efficient interface between computer and telecommunication systems. In developing and coordinating these standards, a considerable amount of effort is expended in initiating and pursuing joint standards development efforts with appropriate technical committees of the Electronics Industries Association, the American National Standards Institute, the International Organization for Standardization, and the International Telegraph and Telephone Consultative Committee of the International Telecommunication Union. This Technical Information Bulletin presents an overview of an effort which is contributing to the development of compatible Federal, national, and international standards in the area of Packed Data Networks and Integrated Services Digital Networks. It has been prepared to inform interested Federal activities of the progress of these efforts. Any comments, inputs or statements of requirements which could assist in the advancement of this work are welcome and should be addressed to:

Office of the Manager
National Communications System
ATTN: NCS-TS
Washington, DC 20305-2010

T A B L E O F C O N T E N T S

T A B L E O F C O N T E N T S
(cont.)

L I S T O F E X H I B I T S

L I S T O F E X H I B I T S
(cont.)

L I S T O F E X H I B I T S
(cont.)

1.0 INTRODUCTION

The Office of the Manager, National Communications System (OMNCS), has been tasked to ensure the provision of National Security and Emergency Preparedness (NS/EP) telecommunications for the Federal Government under all conditions. In support of this task, the OMNCS is working to promote standards within the data communication and telecommunication fields to increase the interoperability of diverse communication systems. These interoperability efforts improve the NS/EP communication capabilities of the nation by providing additional interconnectivity among federal telecommunication assets. One such effort is determining the feasibility of using the excess transmission capacity of the Integrated Services Digital Network (ISDN) out-of-band signaling channels to reconstitute disrupted networks.

1.1 BACKGROUND

Present telecommunication networks use one of many different protocol suites to interconnect their services. Networks employing different protocols cannot communicate with one another because the diverse protocols are incompatible. To address the protocol incompatibility issue, the OMNCS recommended developing mechanisms, such as gateways that provide protocol conversion, to interconnect networks using diverse protocols. These devices will act as translators between the networks while maintaining communications with each network in its native language.

Recent OMNCS efforts have focused on developing a network component to allow X.25 end user traffic to use future ISDNs as transport networks to interconnect disrupted portions of their packet-switched networks. This work investigated the use of the excess transmission capacity of ISDN out-of-band signaling channels to reconstitute disrupted networks. To assess this capability, the two protocols were compared. The comparison was followed by a definition of protocol operating states and transitions among states needed to control and implement an X.25-ISDN gateway. Based on these preliminary operating states, an X.25-ISDN gateway was implemented and tested. The X.25-ISDN gateway and demonstration are described in X.25-ISDN Gateway High Level Language Program Demonstration, National Communications System, May 12, 1989.

Building upon this previous work, an SDL description for a gateway designed to interconnect CCITT X.25 end users, Defense Data Network (DDN) X.25 (TCP/IP) end users, as well as CCITT X.25 and DDN X.25 (TCP/IP) end users, over the ISDN D channel was developed. The specification outlines the internal functionality of gateway components, their modules, and the interrelationship

between these modules. Based on this SDL specification, a multiprotocol gateway was implemented and tested. The Multiprotocol Gateway and demonstration are described in Multiprotocol Gateway Capabilities Demonstration Report, National Communications System, March 2, 1990.

1.2 PURPOSE

The purpose of this document is to present the optimized software component of the multiprotocol gateway, document the gateway software, and discuss potential enhancements to the multiprotocol gateway. This document also includes a user's guide for the gateway operator that describes the gateway menu display and available command options.

1.3 PROJECT REFERENCES

The following documents have been used in the development of this document:

1. X.25/PLP/TCP Gateway Development and Description. National Communications System. February 1986.

2. X.25/PLP-DDN/TCP Gateway SDL Technical Specification. National Communications System. June 1986.

3. X.25 and ISDN Packet Mode State Transition Tables. National Communications System. November 1987.

4. X.25-ISDN Gateway State Transition Tables. National Communications System. May 1988.

5. Data Communications Networks: Interfaces. Recommendations X.20-X.32. CCITT, Vol. VIII - Fascicle VIII.3. 1984.

6. Integrated Services Digital Network (ISDN). Series I Recommendations. CCITT, Vol. III - Fascicle III.5. 1984.

7. X.25-ISDN Gateway SDL Description. National Communications System. August 1988.

8. Functional Specification and Description Language (SDL) Recommendations Z.101 - 104. CCITT, Vol. VI - Fascicle VI.10. 1984.

9. Defense Data Network X.25 Host Interface Specification. Defense Communications Agency. December 1983.

10. Internet Protocol. MIL-STD-1777, Defense Communications Agency, J110. 12 August 1983.

11. Transmission Control Protocol. MIL-STD-1778, Defense Communications Agency. 12 August 1983.

12. X.25-ISDN Gateway High Level Language Program Demonstration. National Communications System. May 12, 1989.

13. Multiprotocol Gateway SDL Description. National Communications System. August 23, 1989.

14. Multiprotocol Gateway Capabilities Demonstration Report. National Communications System. March 2, 1990.

The foundation of the multiprotocol gateway is composed of the CCITT protocol specifications (5 and 6) and the DDN protocol specifications (9 though 11). Other references include documents relating to earlier efforts in support of an X.25-TCP/IP gateway (1 and 2) and efforts involving the X.25-ISDN gateway (3 and 4). The methodology and syntax used to specify the gateway is based on a separate series of CCITT recommendations -- Z series (8). An overview of the SDL is provided below.

1.4 SDL OVERVIEW

The SDL syntax has been developed by CCITT for unambiguously describing the behavior of telecommunication systems. A system is defined in SDL through a set of interconnected blocks. The blocks are connected through unidirectional channels. Each system block describes the functional operation of a portion of the system. Further refinement can be obtained by dividing a block into sub-blocks, which communicate via sub-channels. This successive decomposition is useful for expressing system hierarchy.

In the SDL syntax, each system block contains one or more processes. A process is defined as a communicating finite-state machine composed of multiple processing states. A process is usually used to represent a function of a block, such as setting up or tearing down a connection. Processes communicate through signals, and reception of any one of a predefined set of input signals triggers state transitions within the process.

Processes can also be decomposed into sub-processes to show greater detail. Further refinement can be obtained by partitioning the sub-processes into procedures. The final degree of detail is obtained by splitting the procedures into macros.

In this document, the multiprotocol gateway SDL specification is presented in terms of process diagrams. The process diagrams provide the level of fidelity necessary to

support gateway software development. This is consistent with the I series of recommendations, which also defines the protocols at the process level. The basic symbols used in the process diagrams, including a brief definition of each symbol, are shown in Exhibit 1-1. Descriptive names are given to all SDL symbols that specify the function of the symbol. Additional SDL information can be obtained from reference 8.

1.5 ORGANIZATION

The Multiprotocol Gateway Optimization Report is composed of 6 sections. Section 2 describes the gateway software design. An updated and expanded SDL software specification is included in this section. A listing of the gateway software code is contained in Section 3. Section 4 presents some possible changes/enhancements to the field readiness and usability of this prototype gateway. Section 5 describes the gateway menu display and the various commands available to the gateway operator. Section 6 summarizes the document.

EXHIBIT 1-1

SDL Symbols And Definitions

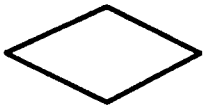
Signal route symbol-- A signal route symbol leads either from one process to another, or from a process to the origin end of a channel, or from the destination end of a channel to a process.

Task symbol-- A task is used to represent operations on data performed on a transition, with the exception of output signal generation and decisions.

Input symbol-- An input symbol attached to a state means that if the signal named within the input symbol arrives while the process is in this state, the transition that follows the input symbol should be interpreted.

Output symbol-- An output symbol represents the sending of a signal from one process to another.

State symbol-- A state is a point in the process where no actions are being performed but where the input queue is monitoring for the arrival of incoming signals.



Decision symbol-- A decision is an action within a transition that asks a question regarding the value of data items available to the process at the instant of executing the action.

Option symbol-- The option symbol is used to describe several alternative process behaviors with one diagram.

Connector symbol-- Any flow line may be broken by a pair of associated connectors, with the flow assumed to be from the out-connector to the in-connector.

Return symbol-- This symbol implies a return to the initial process state.

Stop symbol-- This symbol implies the end of the process.

2.0 GATEWAY SOFTWARE DESIGN

The Multiprotocol Gateway SDL Description (13) provided the foundation from which the gateway implementation was directed. This specification outlined the internal functionality of gateway components, their modules, and the interrelationship between these modules. The gateway software design evolved from this initial SDL description to the gateway design described in The Multiprotocol Gateway Capabilities Demonstration Report (14). An updated and expanded SDL description of the Multiprotocol Gateway, along with a subsystem specification description is contained in this section.

The Multiprotocol Gateway is designed to support the interconnection of CCITT X.25 end users, DDN X.25 (TCP/IP) end users, as well as CCITT X.25 and DDN X.25 (TCP/IP) end users, over the ISDN D channel. Consistent with the mission of the OMNCS, the gateway is designed to support emergency communications. This requires that the gateway be transportable and use hardware that is readily available. For these reasons, the gateway was developed for a DOS-based microcomputer using an INTEL 386 microprocessor. However, the modular design approach to the gateway is applicable to the design of larger gateways, such as a Digital Equipment Corporation MicroVAX II based gateway that could support multiple connections at a faster throughput rate.

2.1 SUBSYSTEM SPECIFICATION

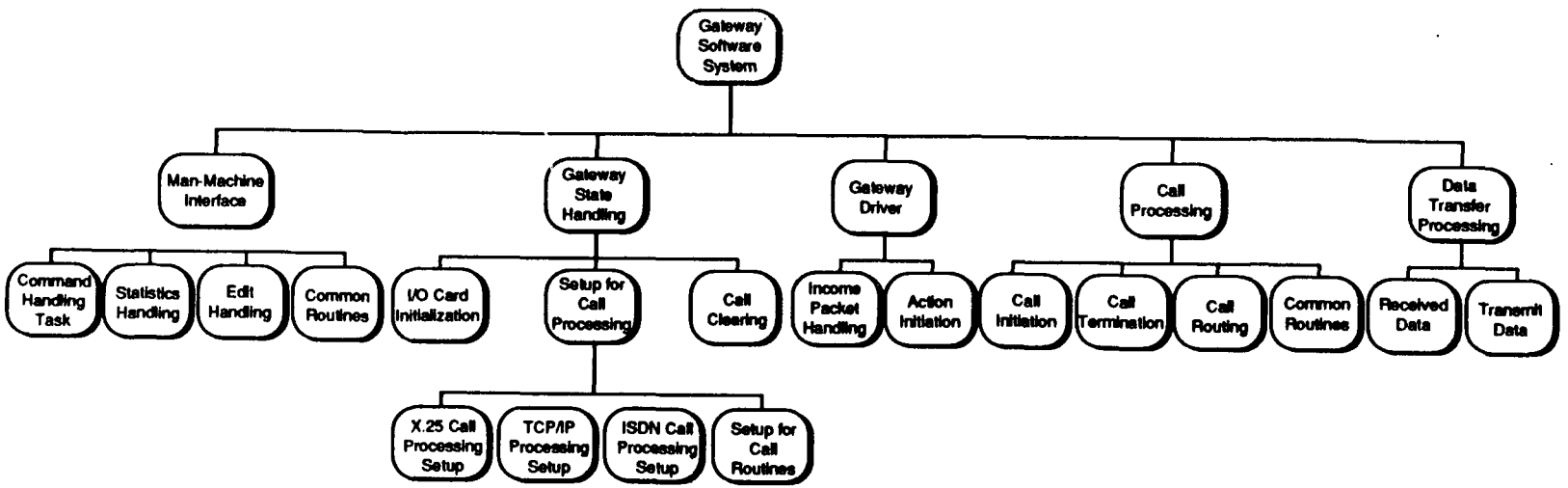
The gateway is structured based on the specific functional processes performed internal to the gateway. A block diagram of the gateway functional processes is shown in Exhibit 2-1. The gateway is composed of five functional modules: the Gateway State Handling Module, Gateway Driver Module, Call Processing Module, Data Transfer Module, and Man-machine Interface (MMI). The Call Processing and Data Transfer modules perform all of the network-specific functions associated with the gateway. The other three modules perform gateway-specific functions. These functional processes are described in more detail below. For further discussion of the gateway software structure see reference 14.

2.1.1 Gateway Driver

Control of all gateway activity is provided through this process. The gateway driver communicates to each of the other processes and controls which processes are activated. Based on information primitives received from the supporting network

EXHIBIT 2-1

GATEWAY BLOCK DIAGRAM



interface cards, the process controls the transitions between operating states. The gateway device driver also tracks the status of the call, gathers call statistics, and monitors the various network interfaces.

2.1.2 Gateway State

This process is responsible for initiating gateway operation. This includes loading the gateway software and downloading the X.25, TCP/IP, and ISDN software. Once the gateway is capable of starting a connection, the gateway driver is notified.

This process is also responsible for resetting the gateway variables and placing the X.25, TCP/IP, and ISDN network modules in a listen state. In the listen state, the network modules wait for a connection attempt from their respective networks and then notify the gateway driver of any attempts.

The Gateway State process performs shutdown procedures upon command from the gateway operator. The link shutdown process controls a sequenced shutdown of all gateway and connection processes.

2.1.3 Call Processing

Upon reception of a connection request packet, the Call Processing process establishes a connection with the specified user on the destination network and returns a connection status message to the gateway driver. This process includes address translation and link establishment.

2.1.4 Data Transfer

After connection establishment, the Data Transfer process accepts data packets from the respective network module and forwards the data to the destination network output buffer for transmission to the destination end user. The Data Transfer process also includes the repacketization and connection termination functions.

2.1.5 Man-Machine Interface

The MMI is responsible for the processing involved in presenting a user-friendly interface to accommodate the following three information flows:

- Commands flowing from the operator to the gateway. The operator is a passive operator. No commands are provided to alter the network data in any form. This

flow also includes feedback to indicate successful or unsuccessful command completion.

- Routing information flowing from the operator. This process performs all processing necessary to allow easy entry of routing information. Changes made to the routing information during gateway operation will not have full effect until the next gateway session.

- Performance information from the gateway to the operator. This information includes statistical and trace information. The process performs the necessary action to present the requested information in a usable form.

2.2 PROTOCOL DRIVER SUITES

The Protocol Driver Suites include the ISDN Protocol Suite, the X.25 Protocol Suite, and the TCP/IP Protocol Suite. These are described below.

2.2.1 ISDN Protocol Suite

This process performs all physical, data link, and network layer operations specified for an ISDN basic rate interface (BRI). The physical layer covers the physical link between devices and the rules by which bits are passed from one to another. The data link layer attempts to make the physical link reliable and provides the means to activate, maintain, and deactivate the link. The network layer provides for the transparent transfer of data between transport entities. The process receives information packets from the ISDN and after completing the required processing, forwards the appropriate information to the gateway device driver for further processing by the gateway. In addition, information primitives sent from the gateway are directed to the ISDN via the ISDN protocol suite process.

2.2.2 X.25 Protocol Suite

This process is analogous to the ISDN Protocol Suite process with respect to the X.25 protocol.

2.2.3 TCP/IP Protocol Suite

This process performs all physical, data link, network layer, and transport layer operations specified for a DDN (TCP/IP) X.25 user. The transport layer (TCP) ensures that data units are delivered error free, in sequence, with no losses or duplications. This process is analogous to the X.25 Protocol

Suite with respect to DDN X.25 and the TCP/IP protocol.

2.3 SYSTEM USERS

Three types of users are defined in this multiprotocol gateway specification. One type of user is a real-time X.25 or DDN end user making use of the additional transport facilities offered by the gateway. The second type of user is the passive run-time operator for the gateway. A third user type is one or more remote gateways that are in the process of forwarding information received from remote X.25 or TCP/IP end users.

The real-time subscriber uses the gateway's capabilities when access across the user's principal transport mechanism has been disrupted. The gateway provides connectivity through the excess capacity of existing ISDNs. The real-time subscriber can obtain access to the gateway's capabilities through one of two modes of connection. The first mode is through a direct, hard-wire DTE-to-DCE connection to the gateway. In the second mode, the gateway is a defined DTE endpoint on a network (X.25 or DDN) and an end user can access the gateway by routing a call to the gateway through the operational portion of the associated network.

The gateway described in this specification does not support protocol translation for session level and above. The applications used by each of the end users (X.25 to TCP/IP) during a call must be compatible. The gateway does not modify the protocol data units above the transport level in any way. As a next step in the multiprotocol development, an application protocol translation capability (e.g., FTP-FTAM, SMTP-X.400, TELNET-Virtual Terminal Protocol) could be incorporated into the gateway.

The second type of system user is the gateway operator. The operator controls all gateway functionality including startup, shutdown, and status monitoring. In addition, the gateway operator controls the loading and maintenance of the addressing and routing table information.

When working together to provide the services specified above, a local and remote gateway form a master/slave relationship. Address translation and call typing is performed at the local (source) gateway. This allows the remote gateway to process the call without regard to the source location and provides the ability to configure multiple (more than two gateways) gateway bridges. The source gateway can be viewed as just another user type in that, much as the X.25 and DDN type end users, it is simply using the resources in the remote gateway to complete its call.

2.4 GATEWAY SDL SPECIFICATION

The functional operation of the Multiprotocol Gateway is described by the SDL diagrams contained in this section. This SDL description is an updated and expanded version of the Multiprotocol Gateway SDL description given in reference 13. This SDL description reflects the optimized gateway design based on the preliminary design and testing phase of the gateway development.

EXHIBIT 2-2

Gateway System SDL State Diagram

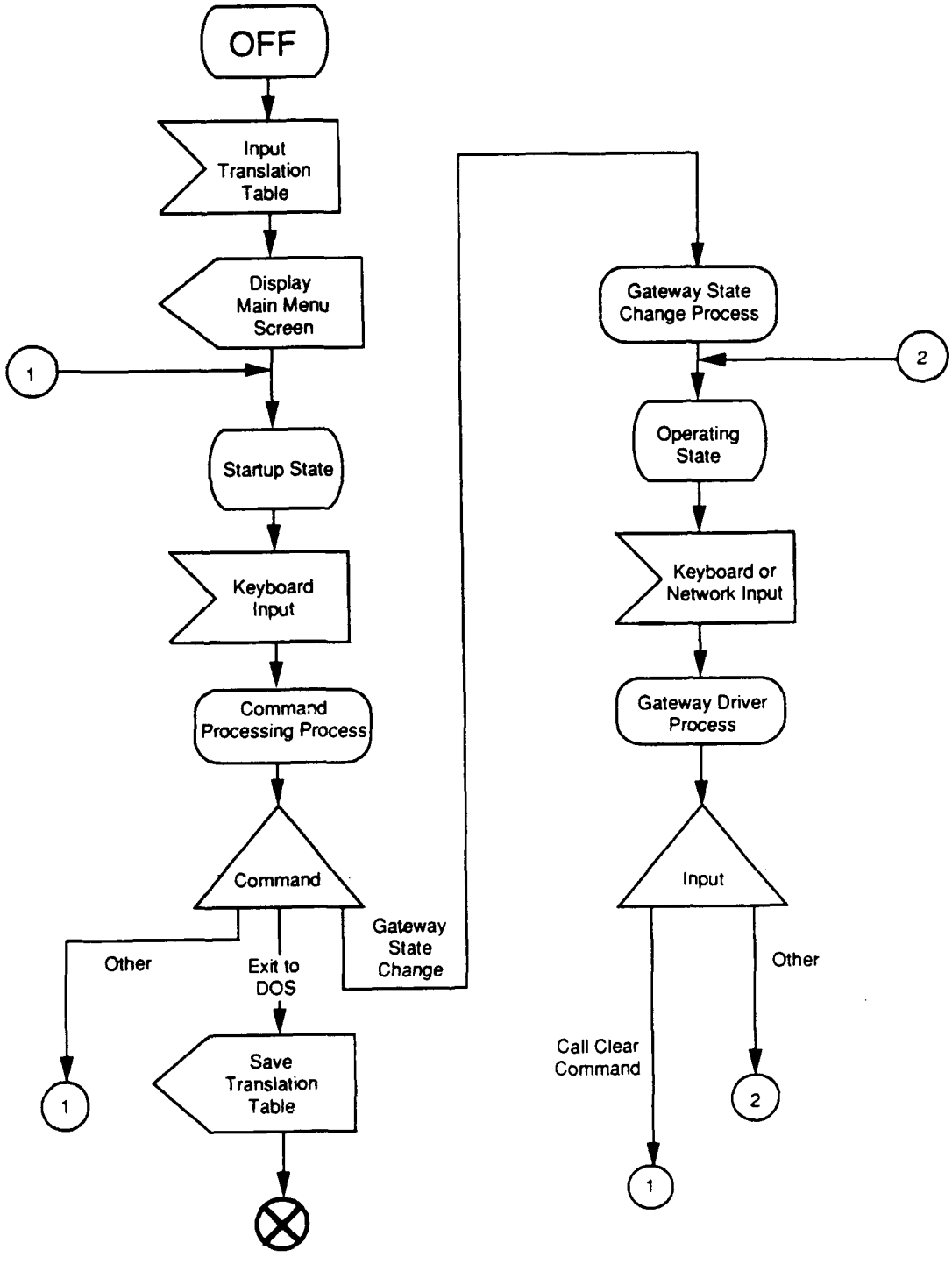


EXHIBIT 2-3

Command Processing Process

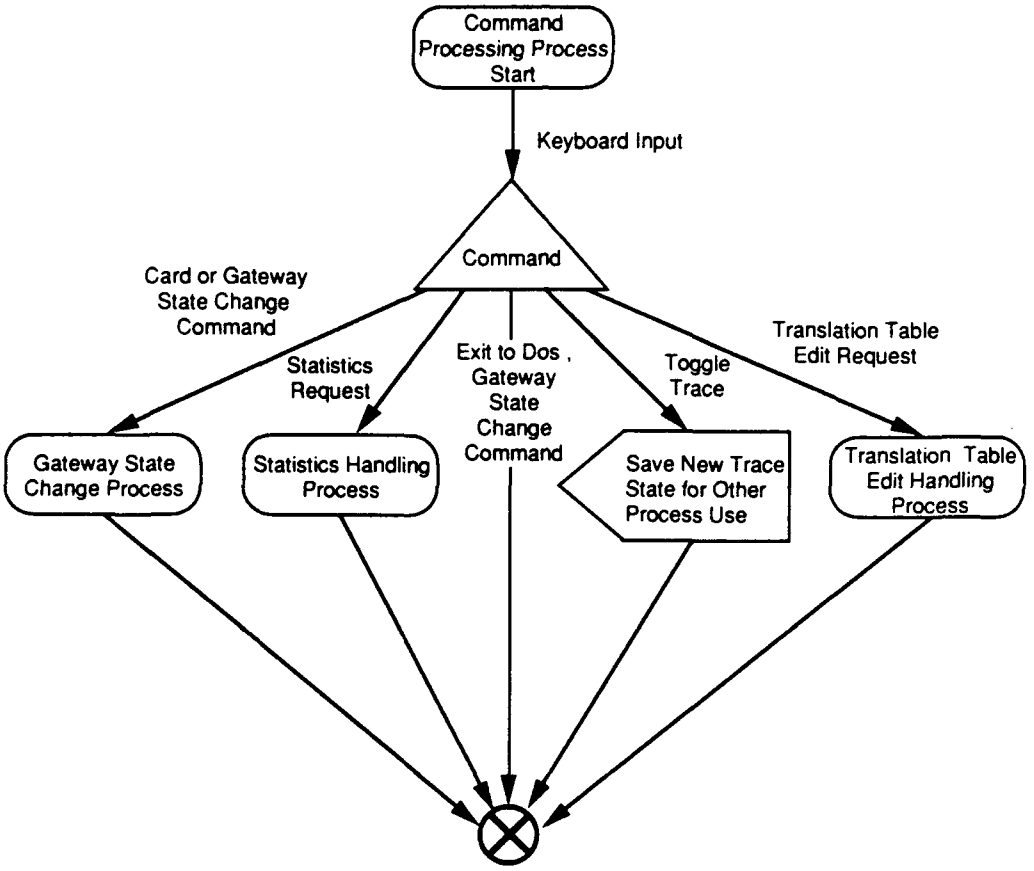


EXHIBIT 2-4

Gateway States Change Process

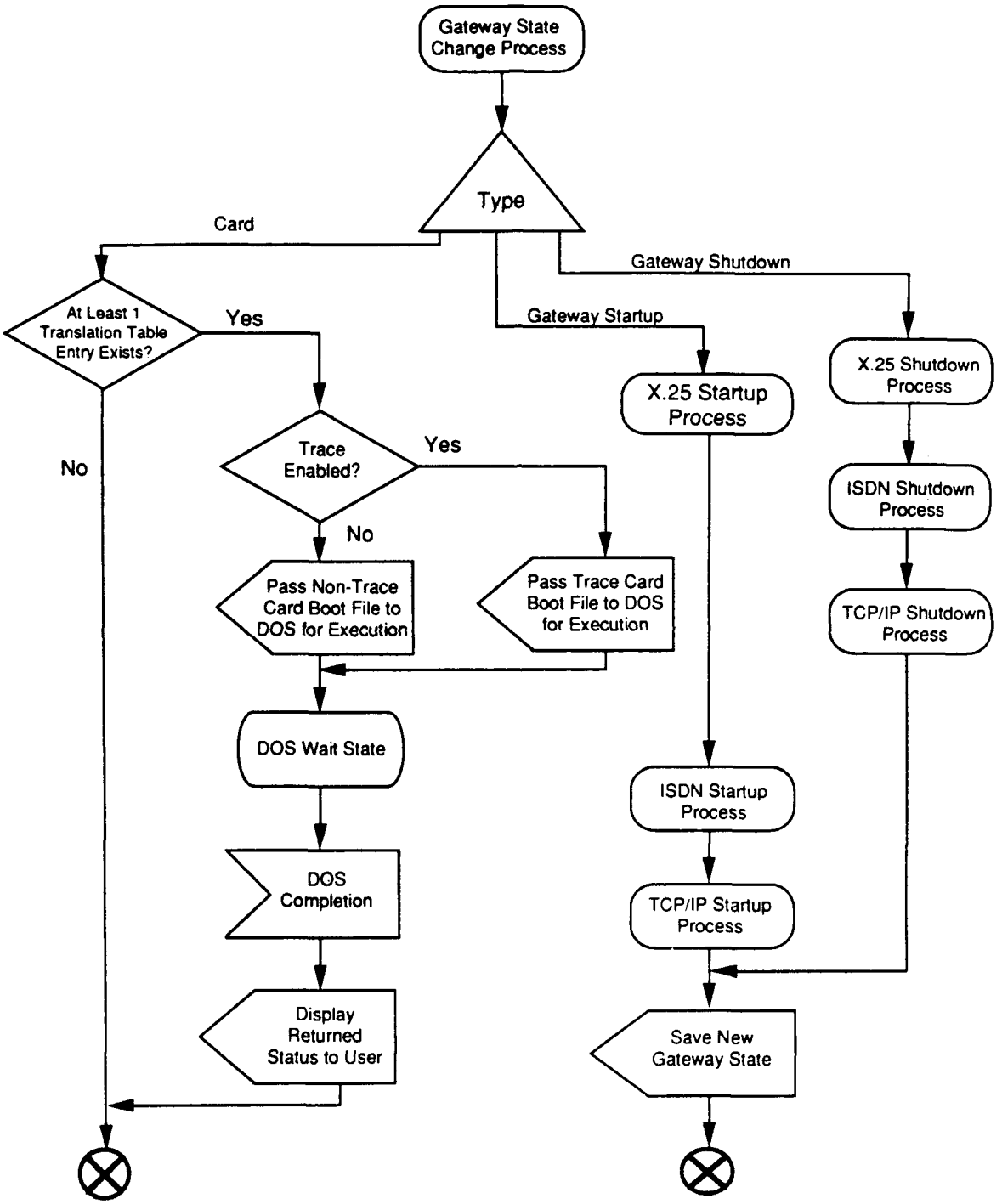


EXHIBIT 2-5

Gateway Driver Process

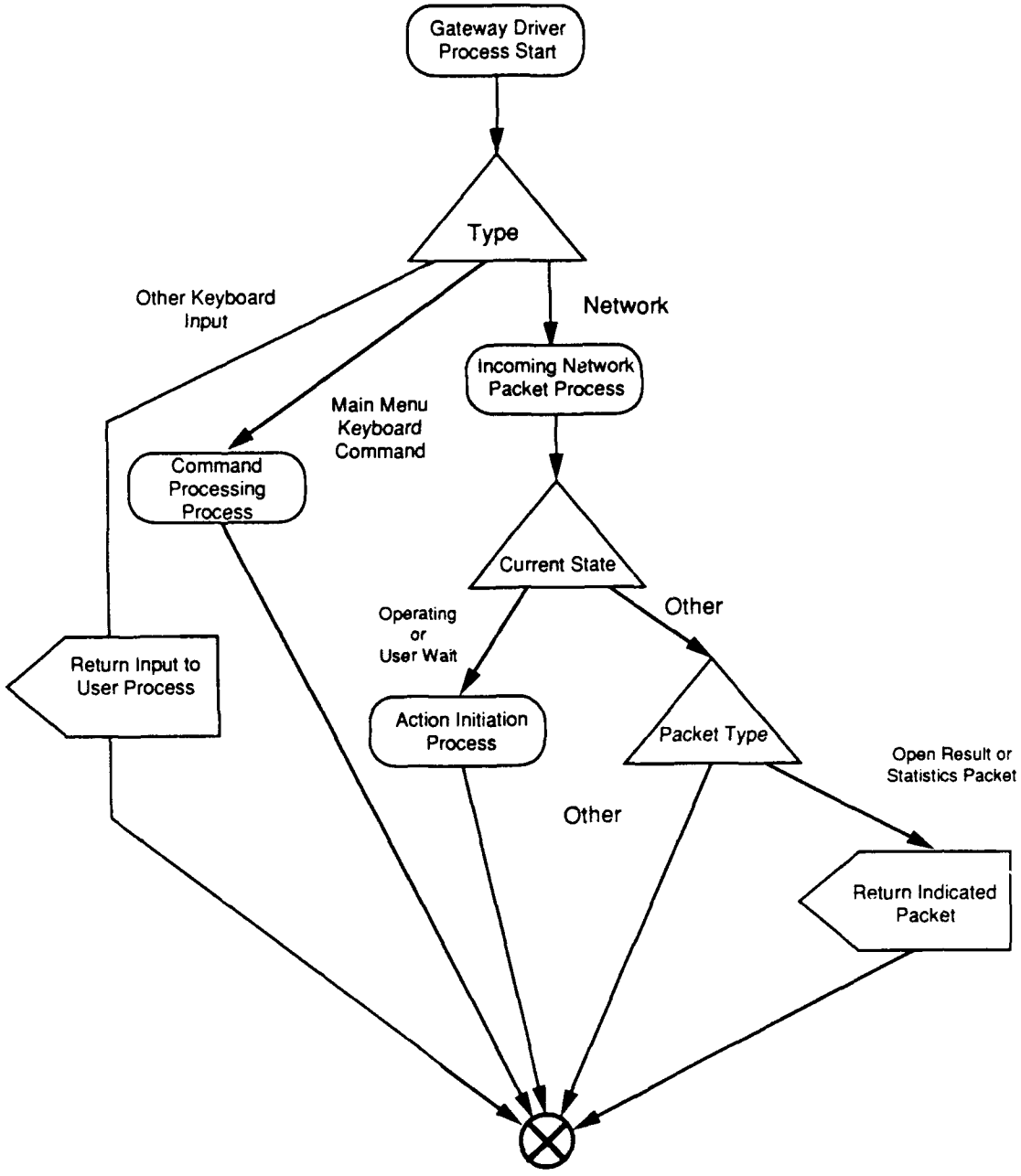


EXHIBIT 2-6

Statistics Handling Process

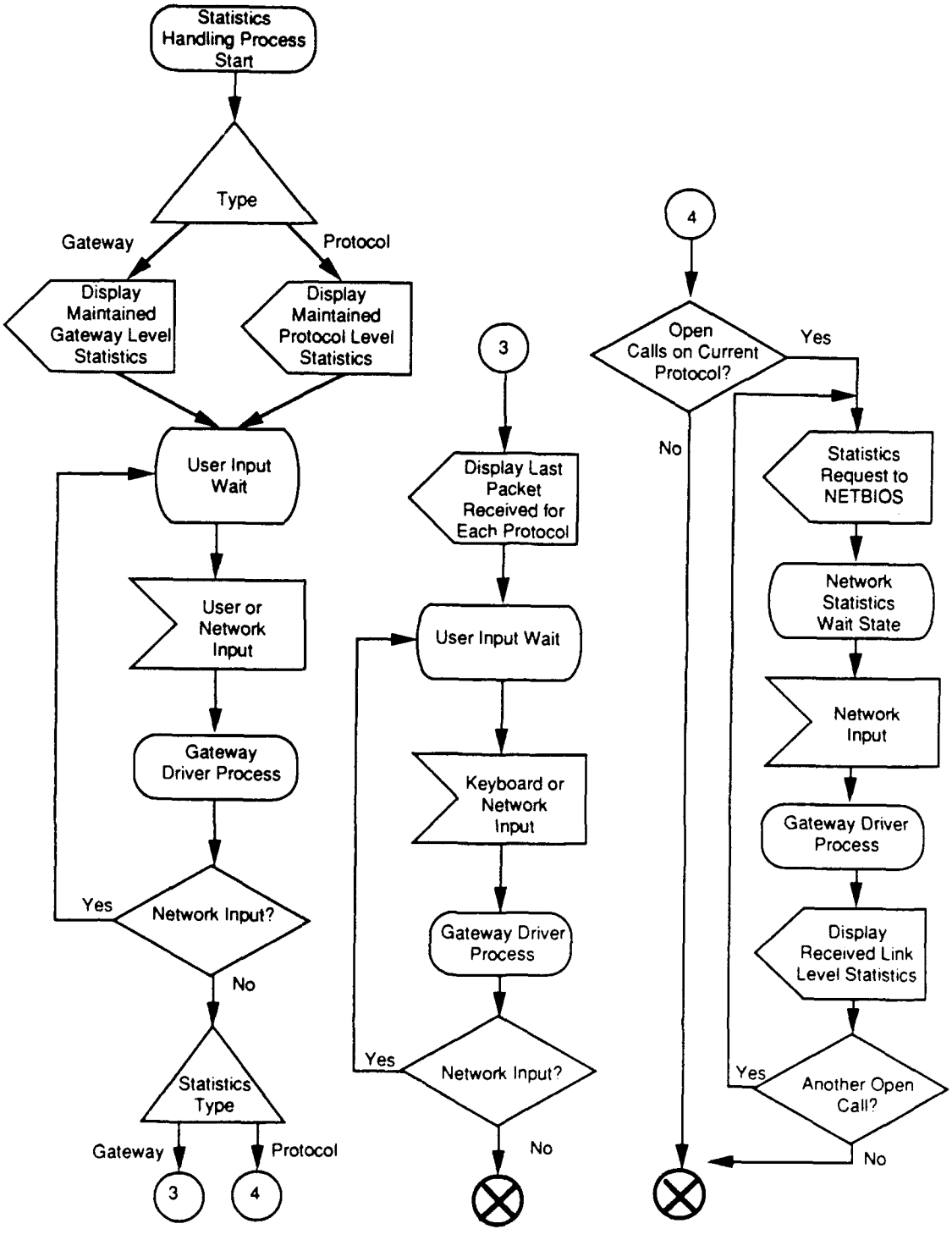


EXHIBIT 2-7

Edit Handling Process

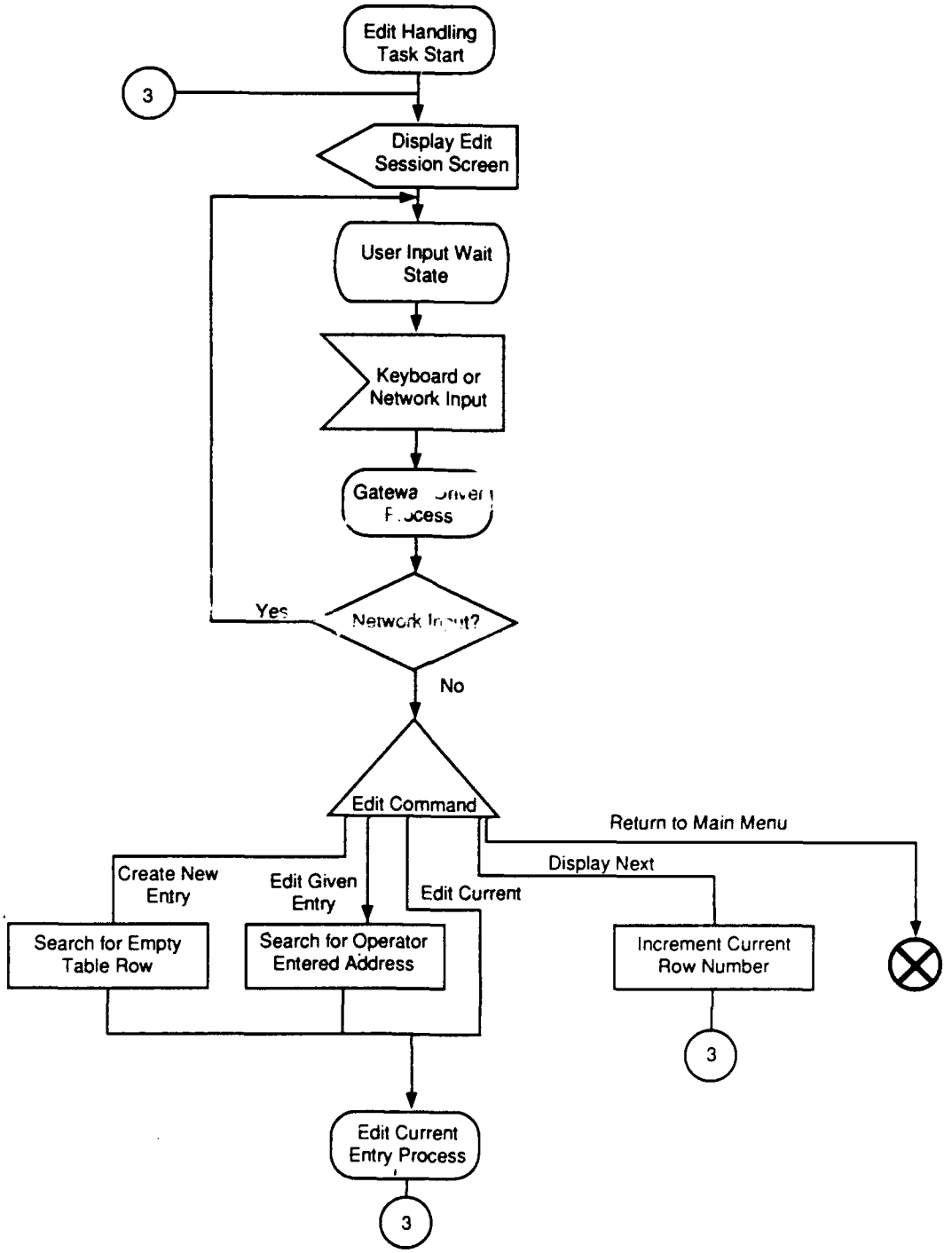


EXHIBIT 2-8

X.25 Startup Process

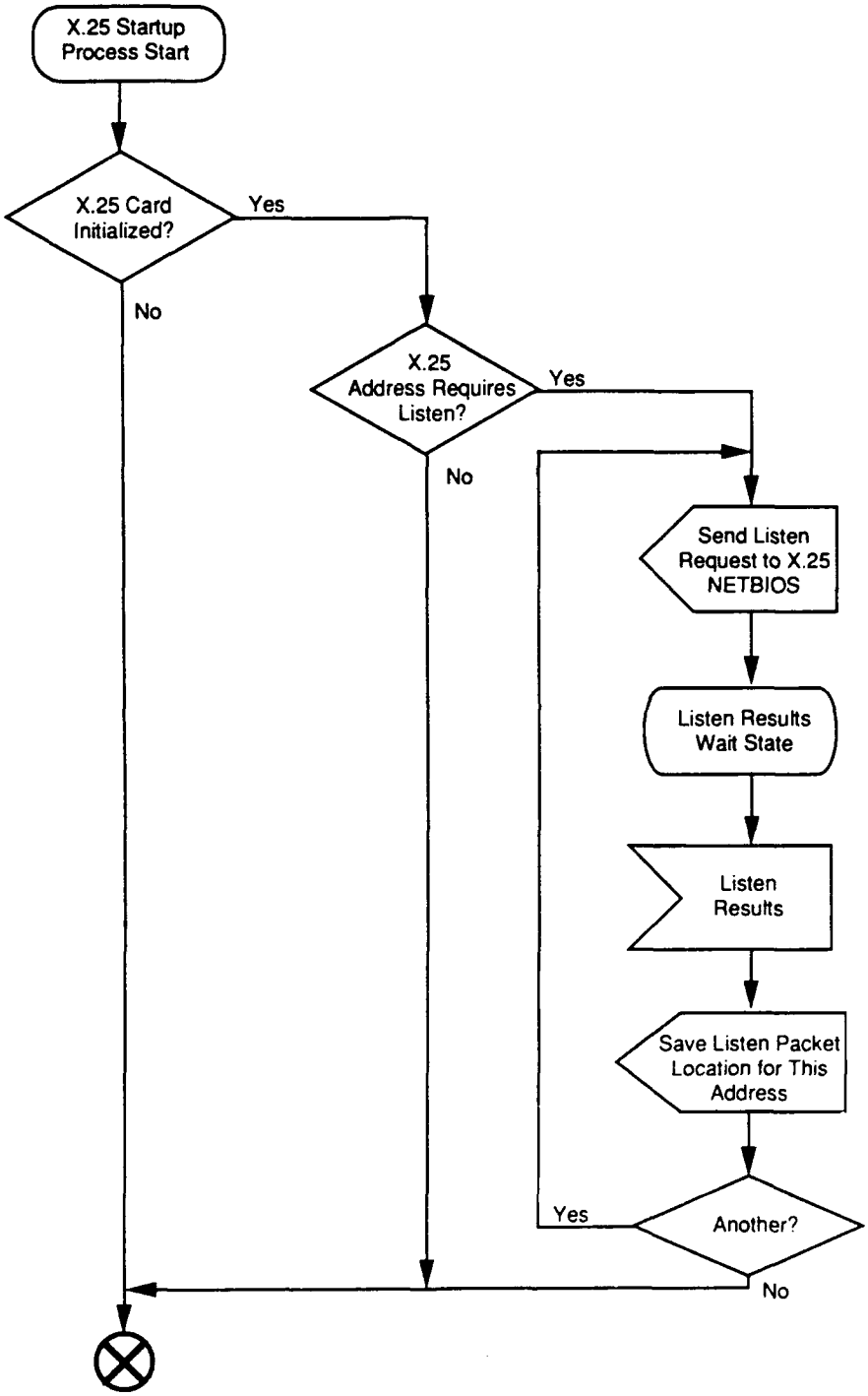


EXHIBIT 2-9

TCP Startup Process

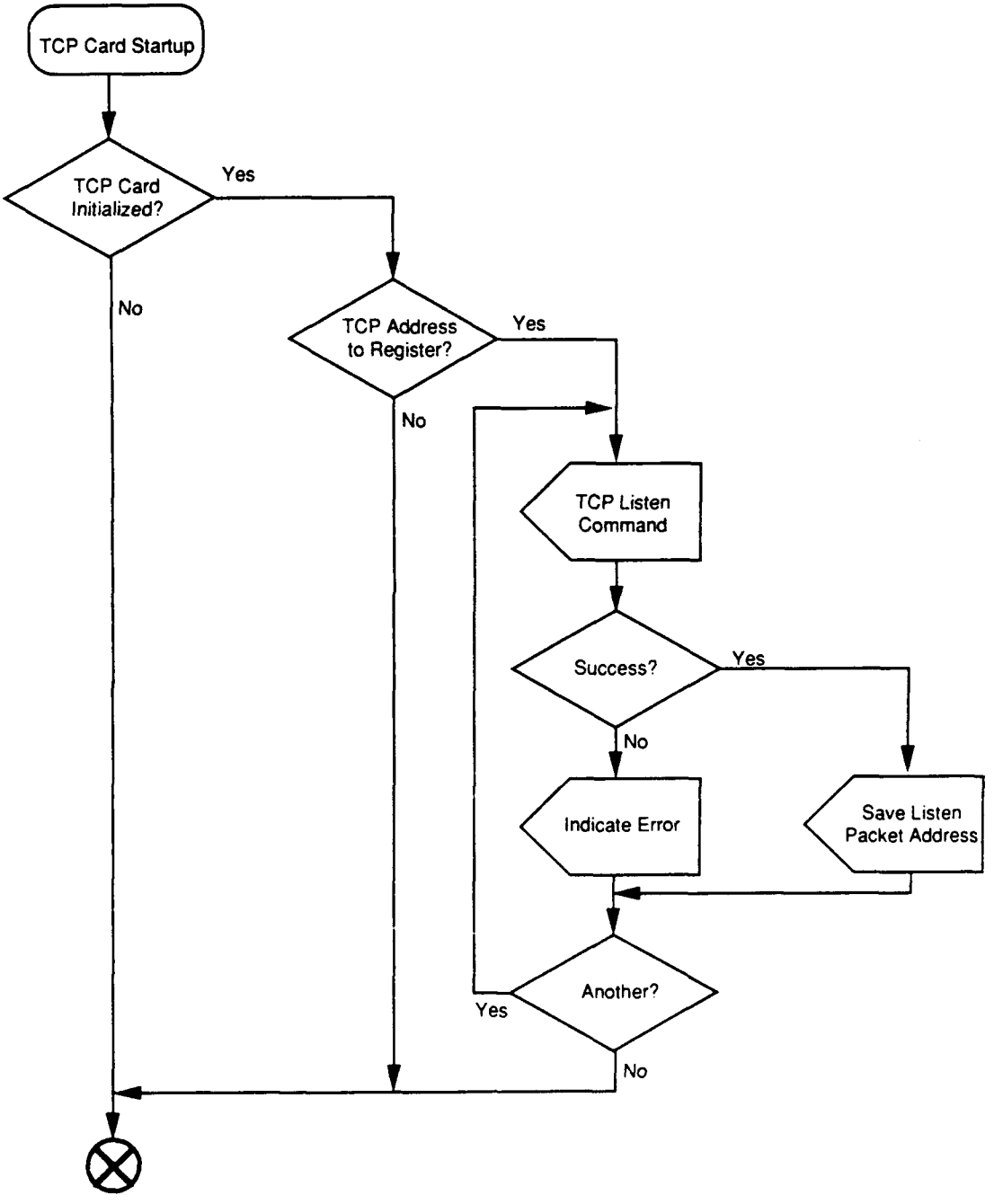


EXHIBIT 2-10

ISDN Startup Process

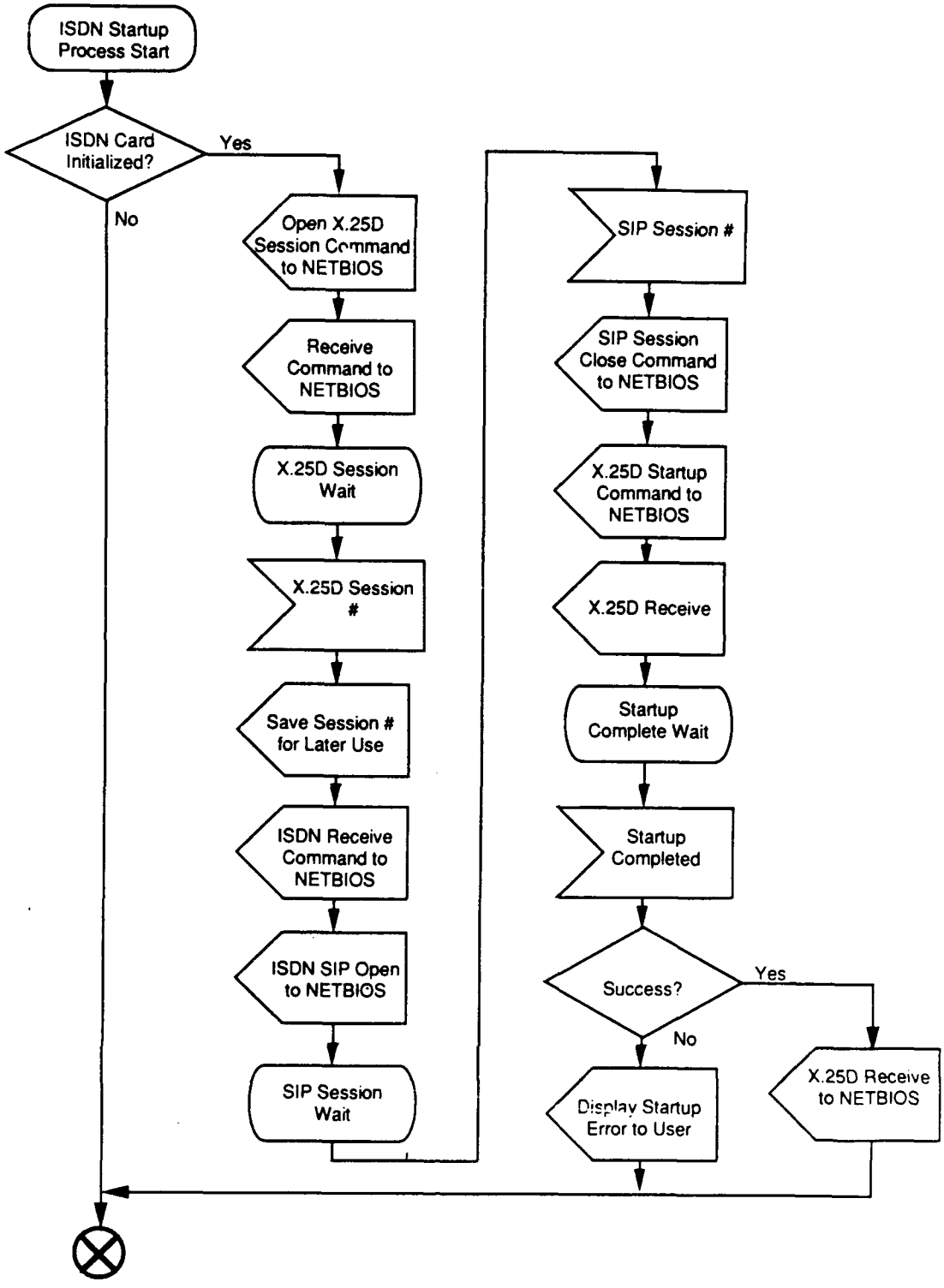


EXHIBIT 2-11

X.25 Shutdown Process

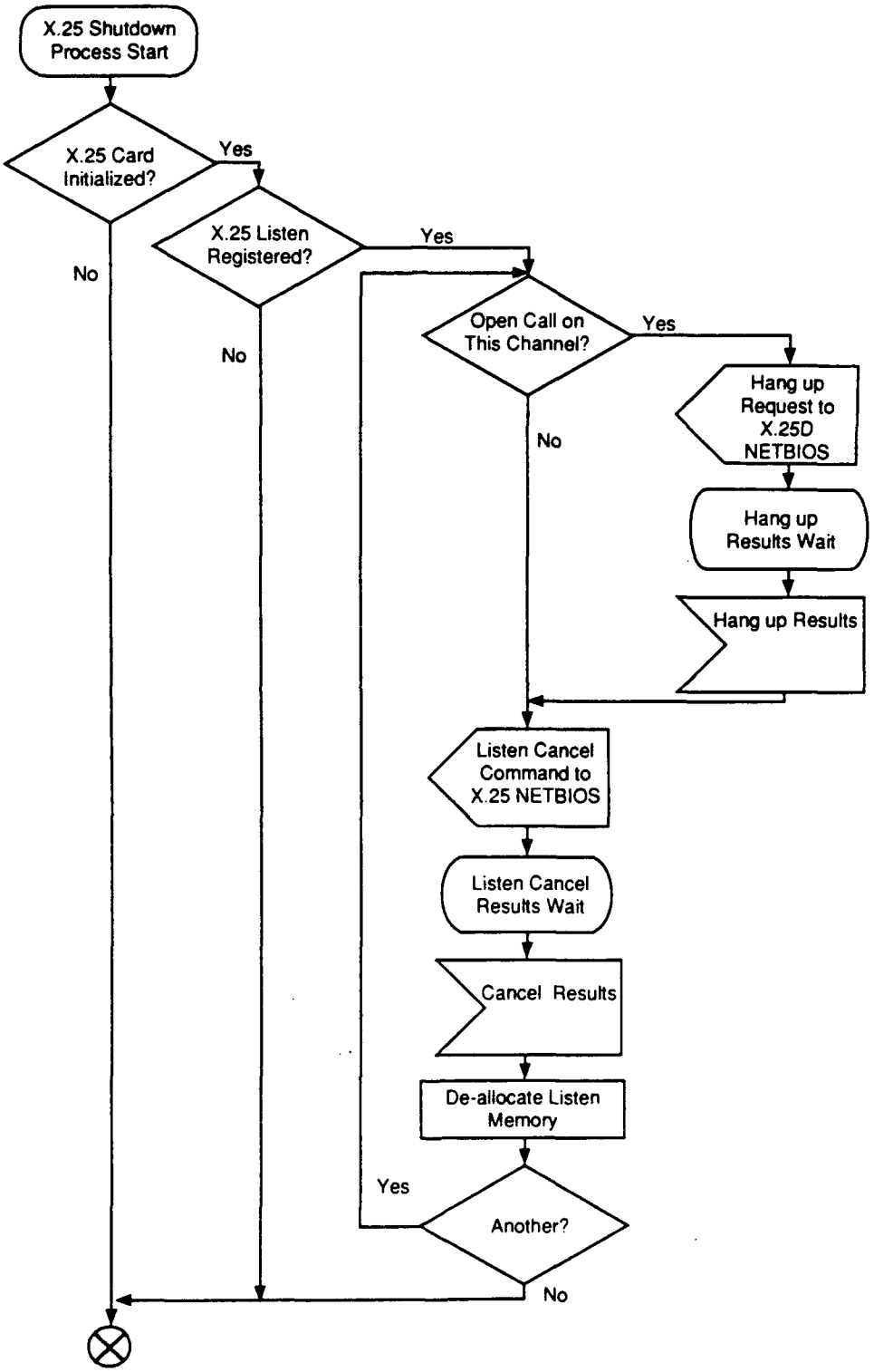


EXHIBIT 2-12

TCP Shutdown Process

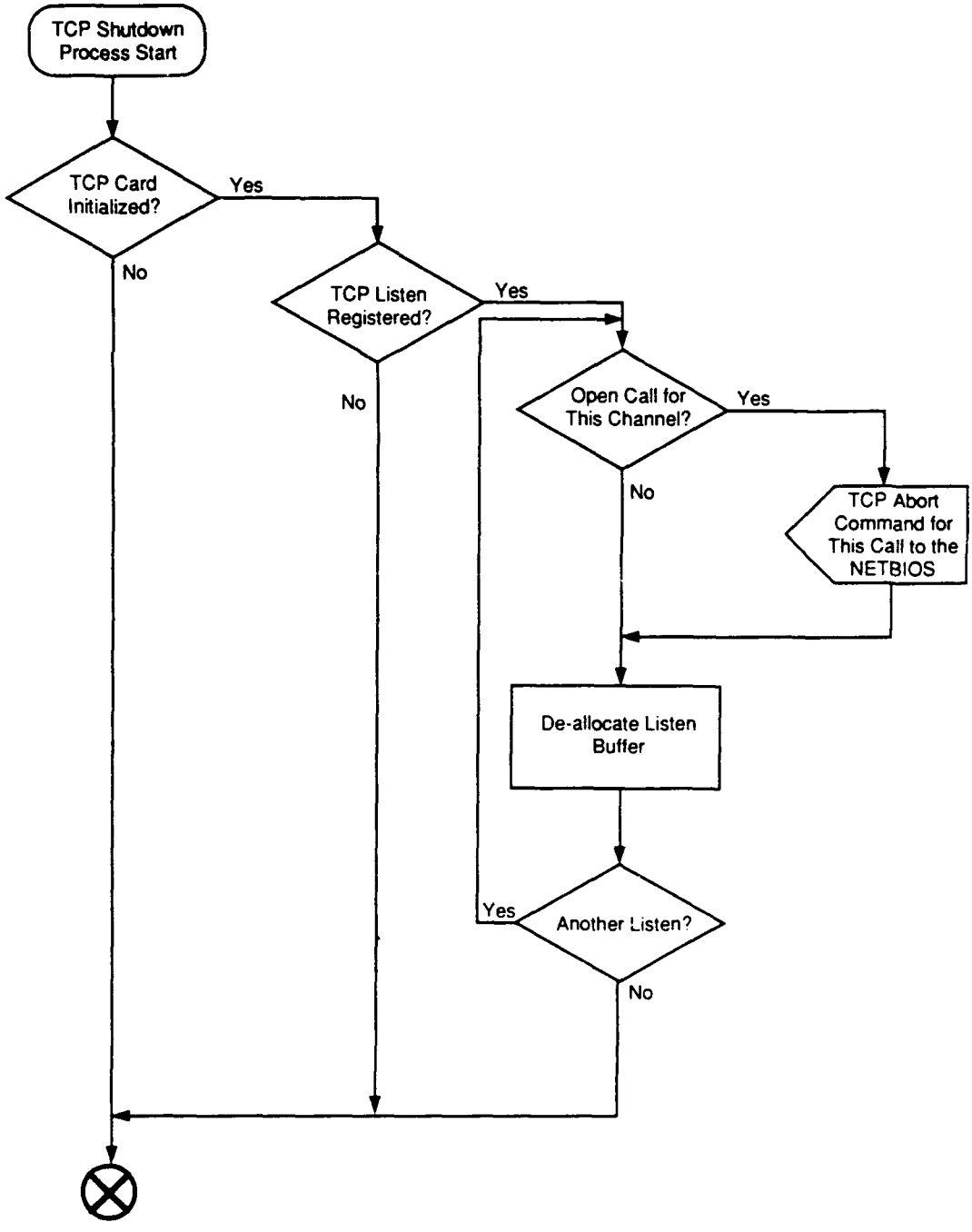


EXHIBIT 2-13

ISDN Shutdown Process

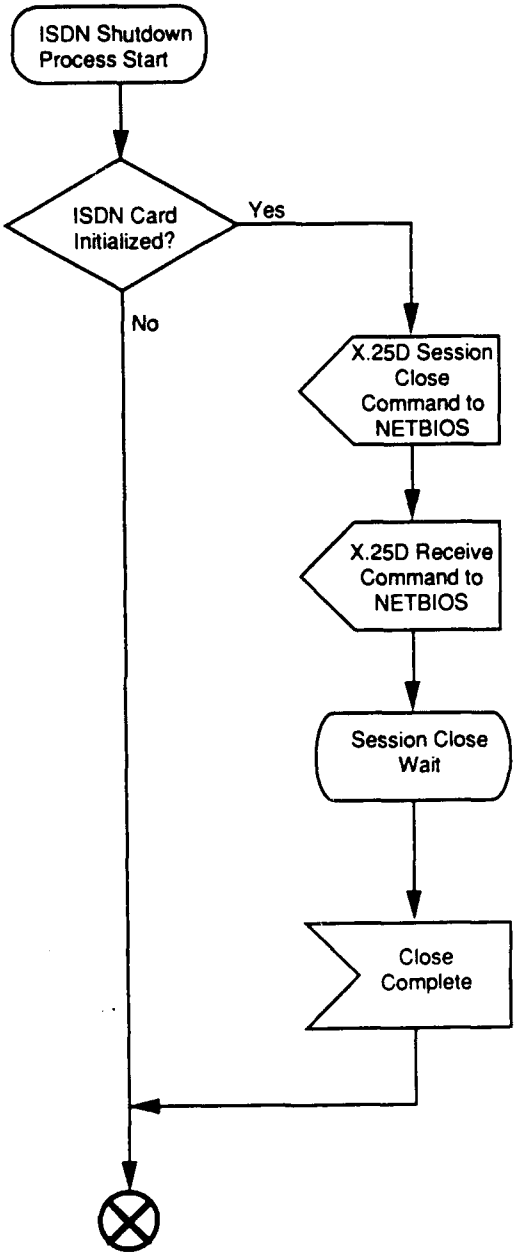


EXHIBIT 2-14

Incoming Network Packet Process

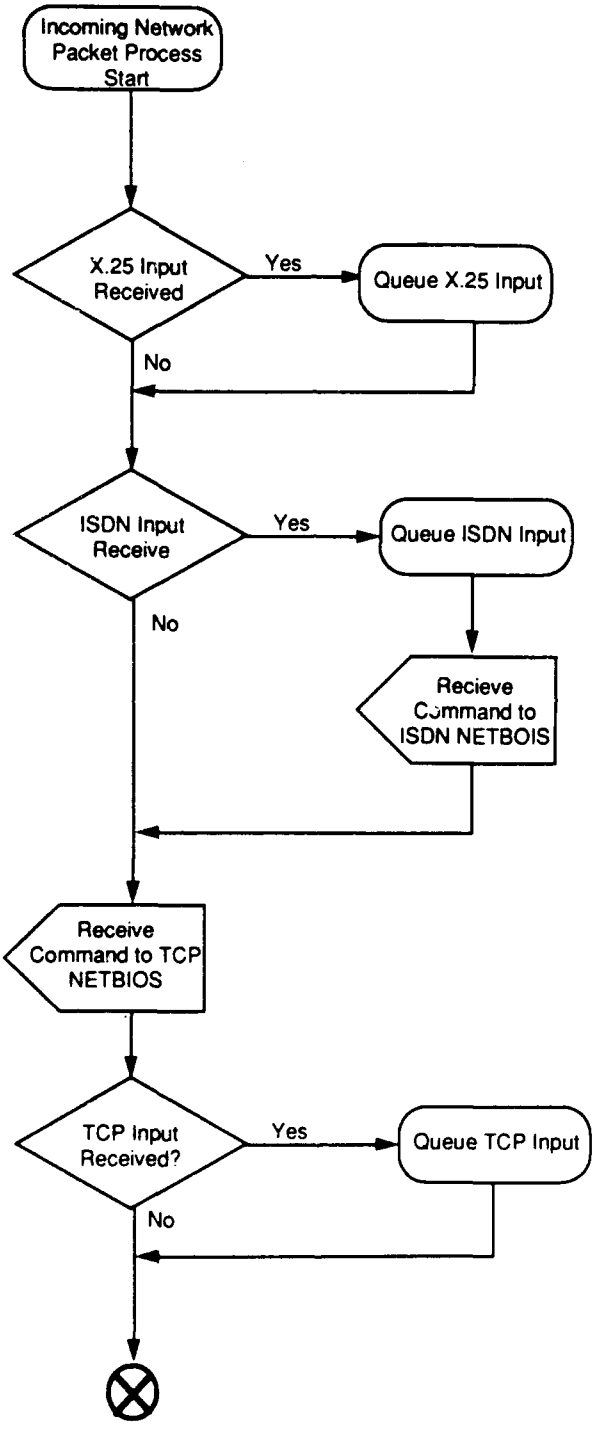


EXHIBIT 2-15

Action Initiation Process

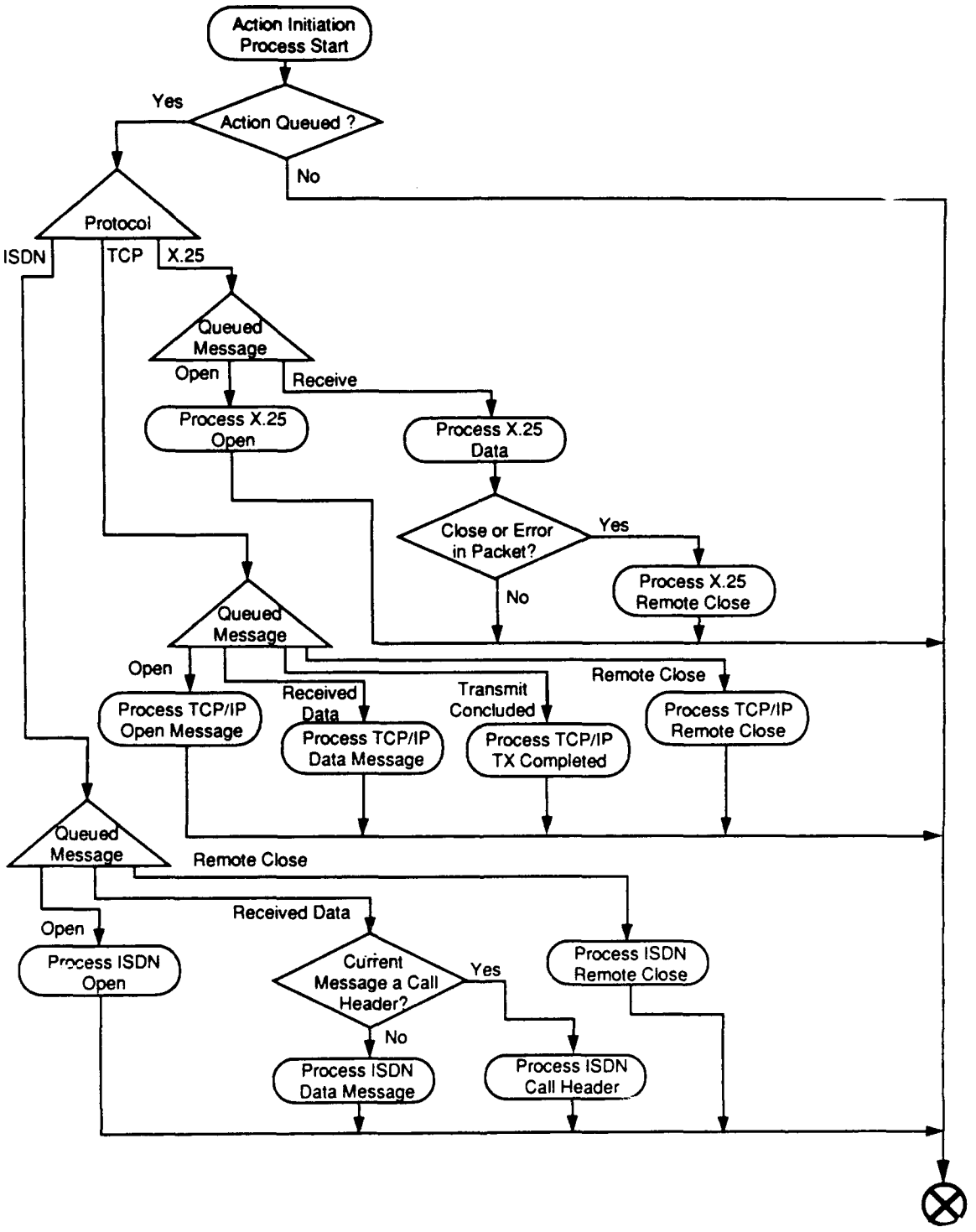


EXHIBIT 2-16

Edit Entry Process

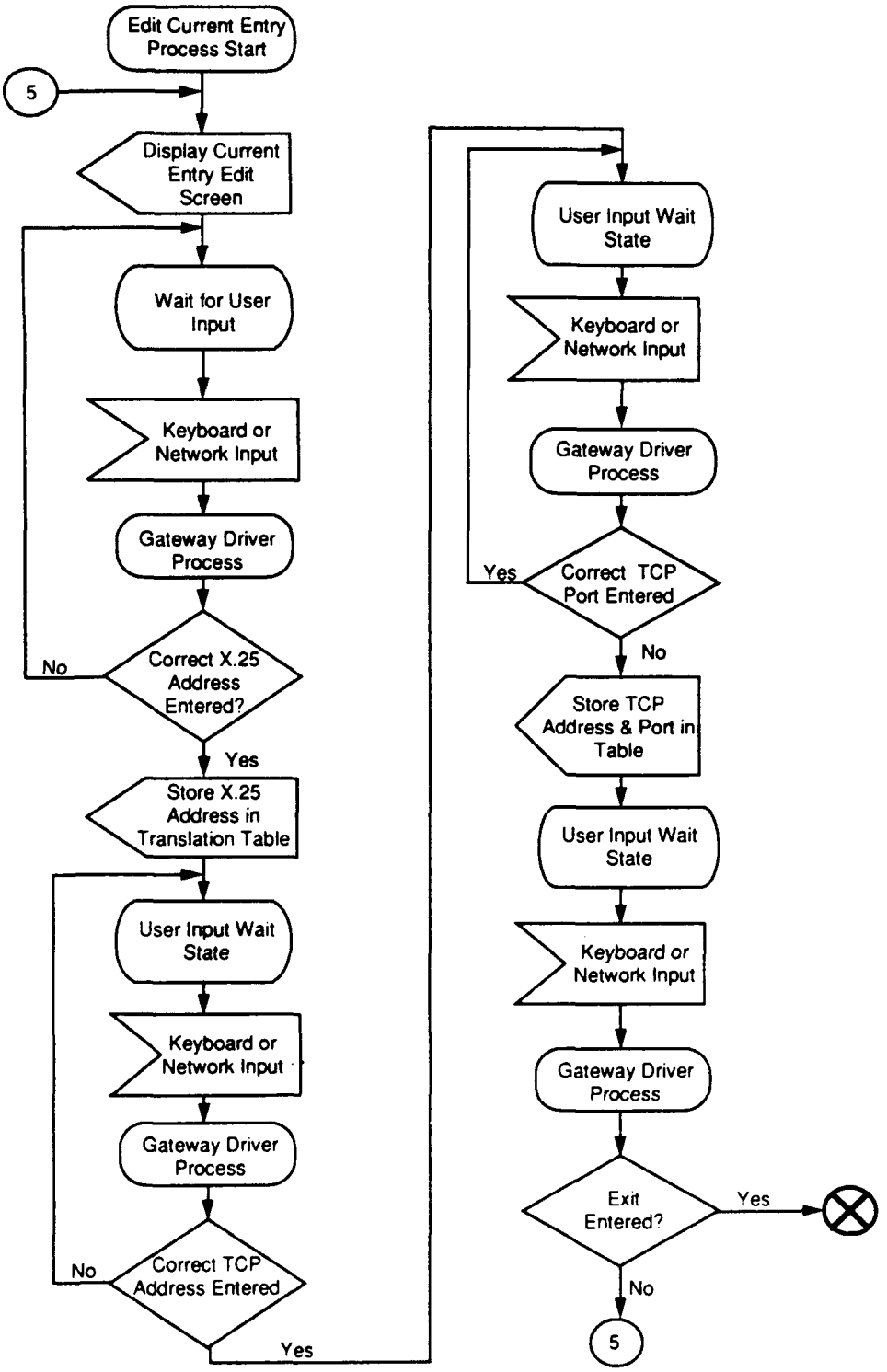


EXHIBIT 2-17

Queue X.25 Input Process

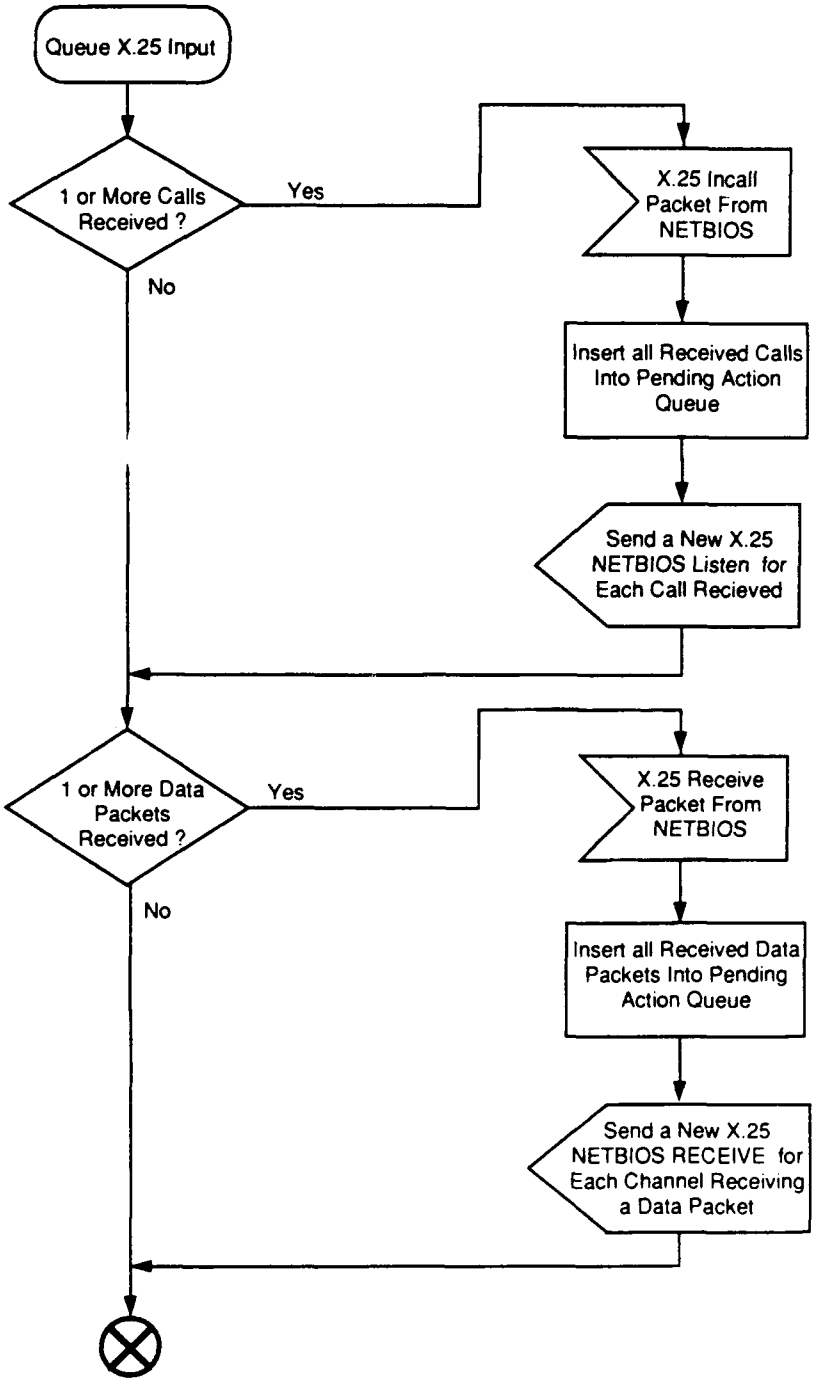


EXHIBIT 2-18

Queue ISDN Input Process

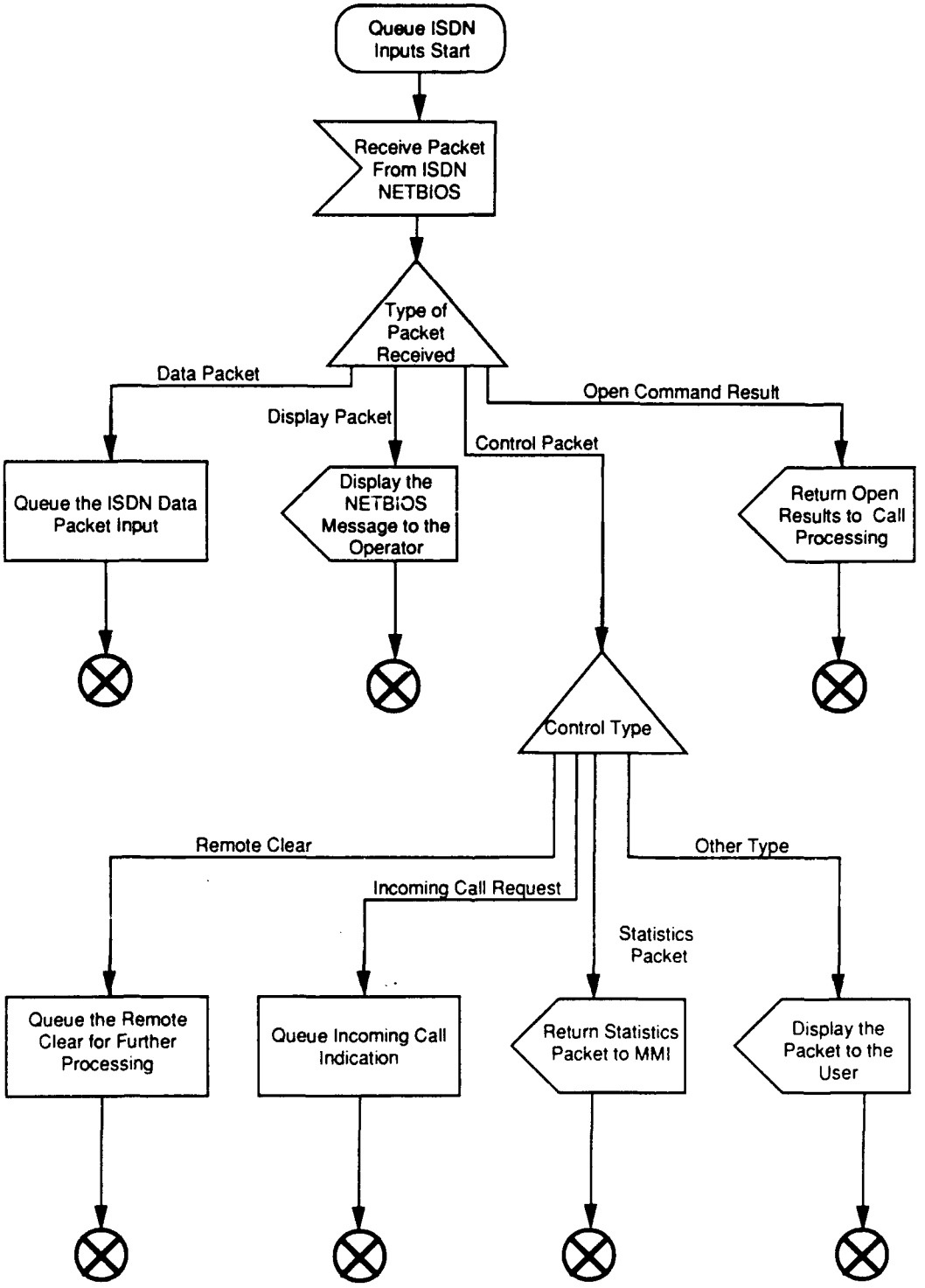


EXHIBIT 2-19

Queue TCP Input Process

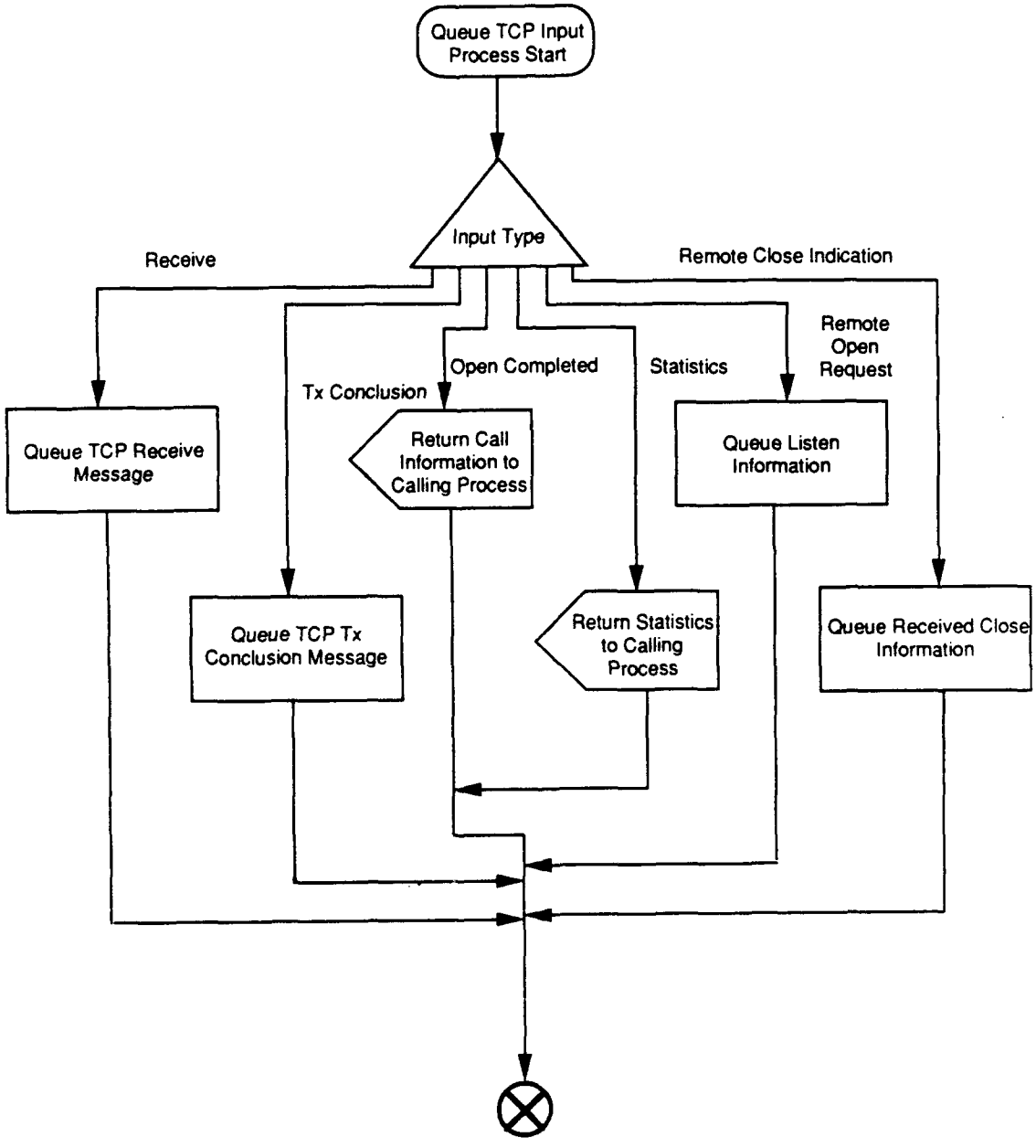


EXHIBIT 2-20

Process ISDN Open

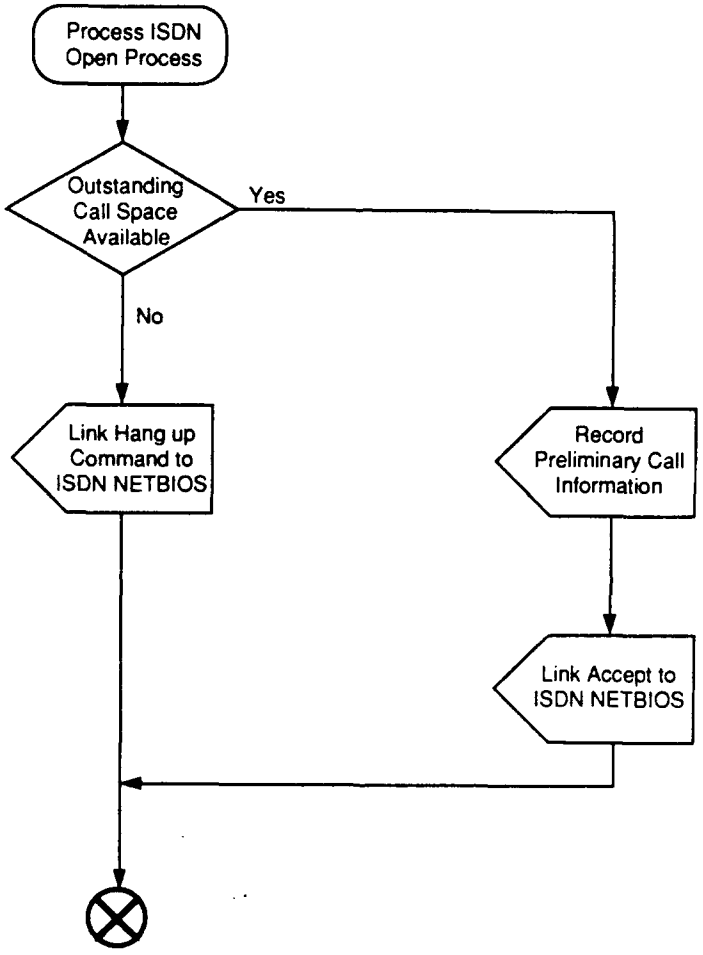


EXHIBIT 2-21

Process X.25 Open

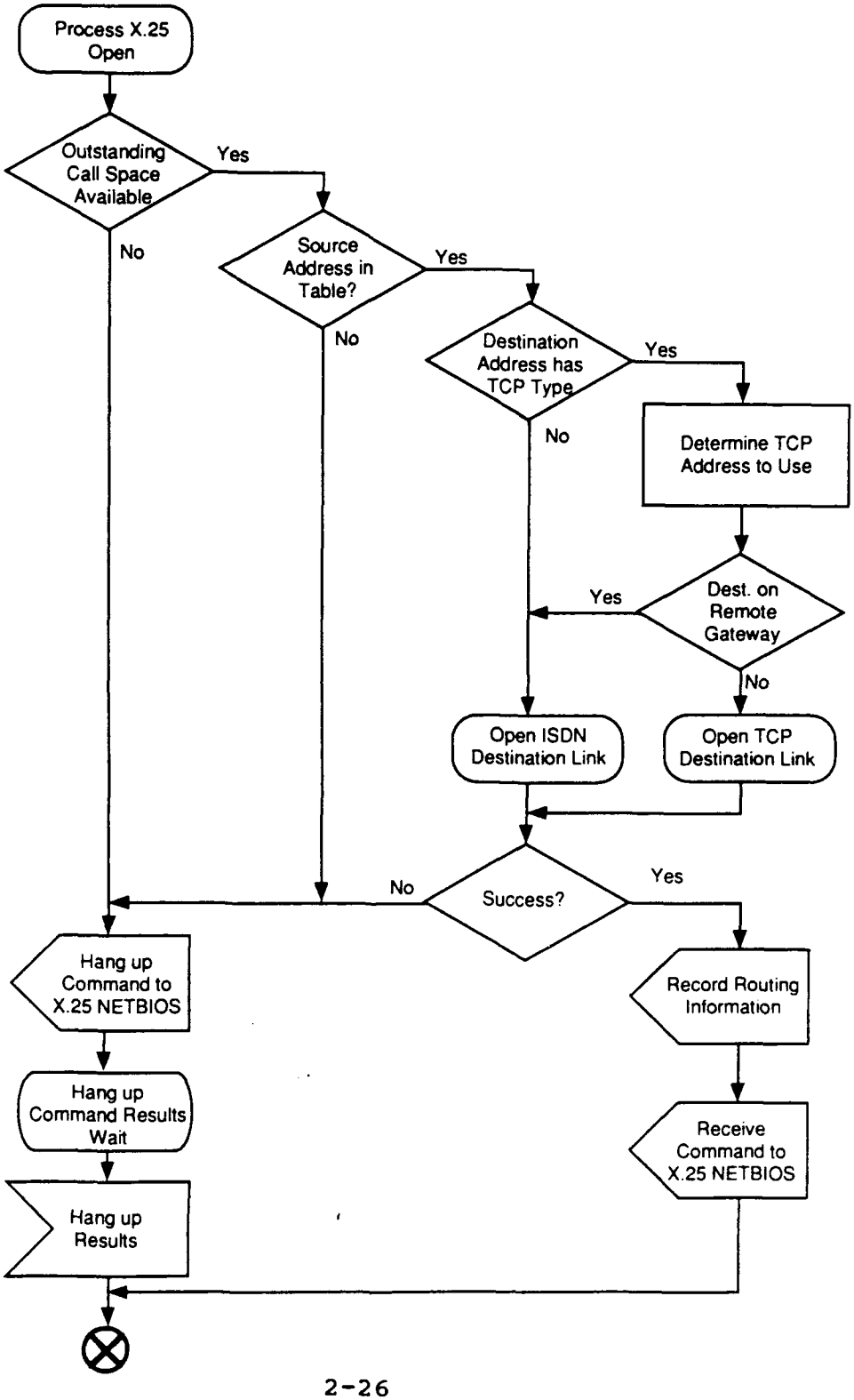


EXHIBIT 2-22

Process TCP Open

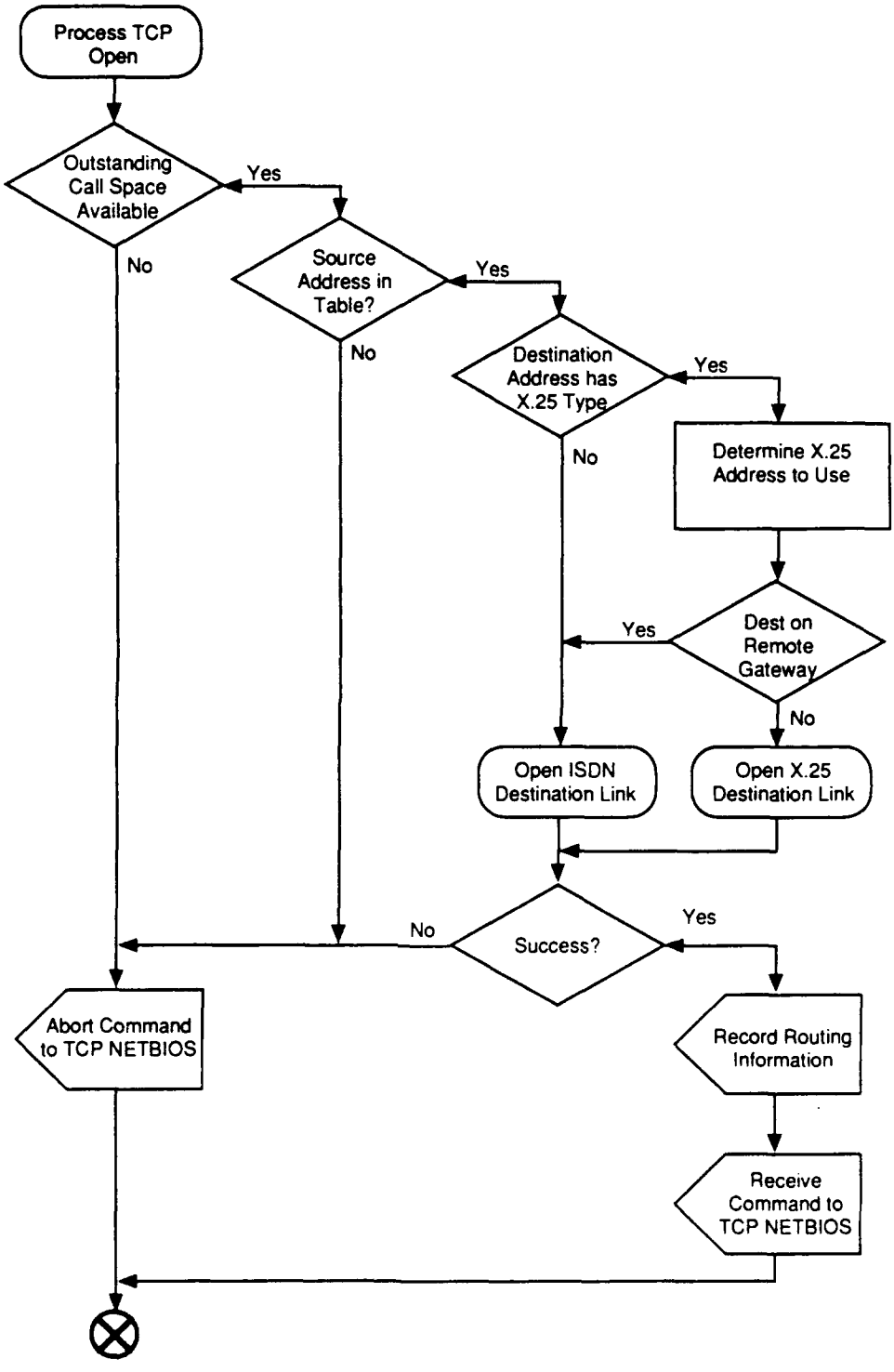


EXHIBIT 2-23

Process X.25 Data

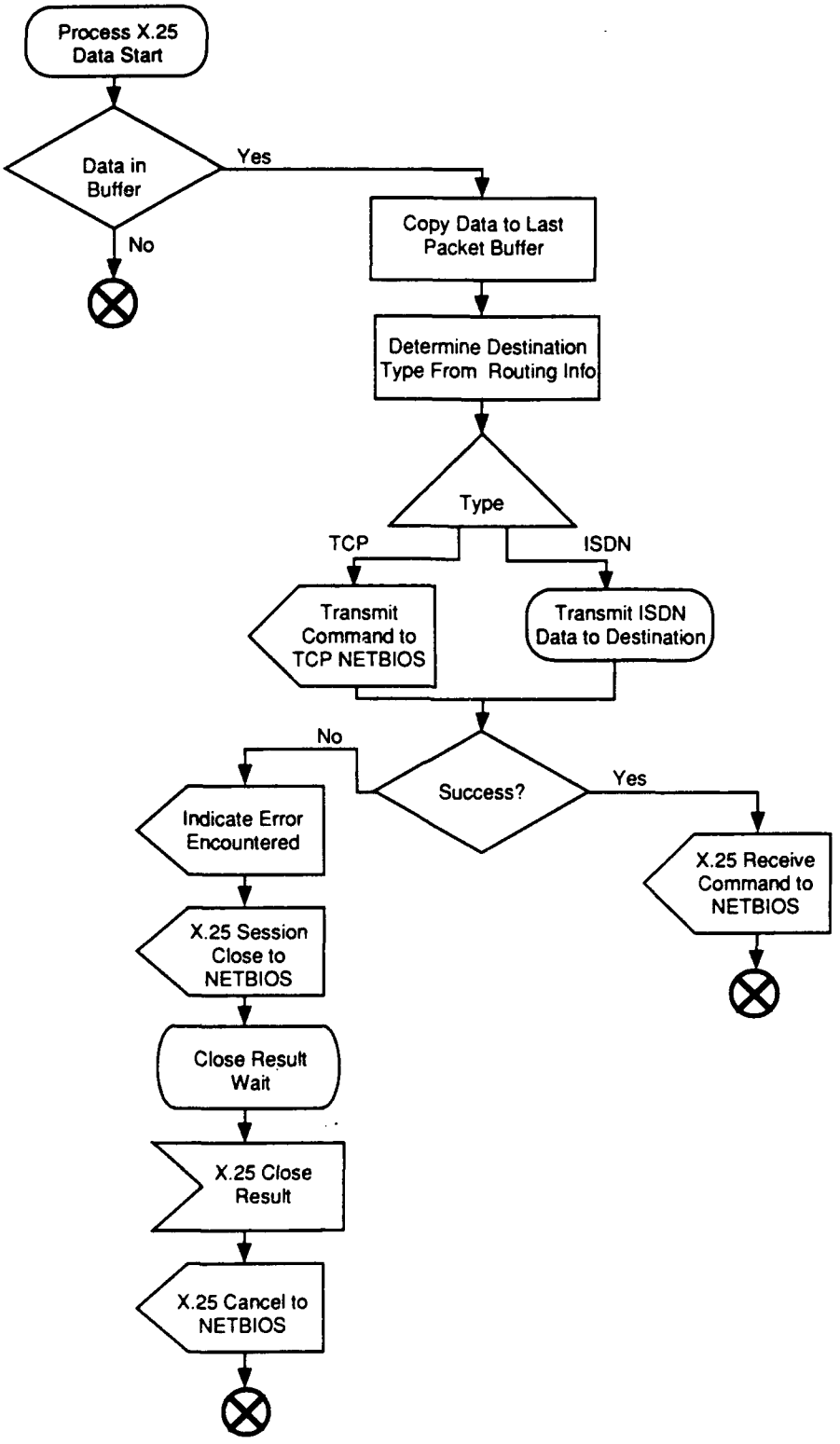


EXHIBIT 2-24

Process ISDN Data

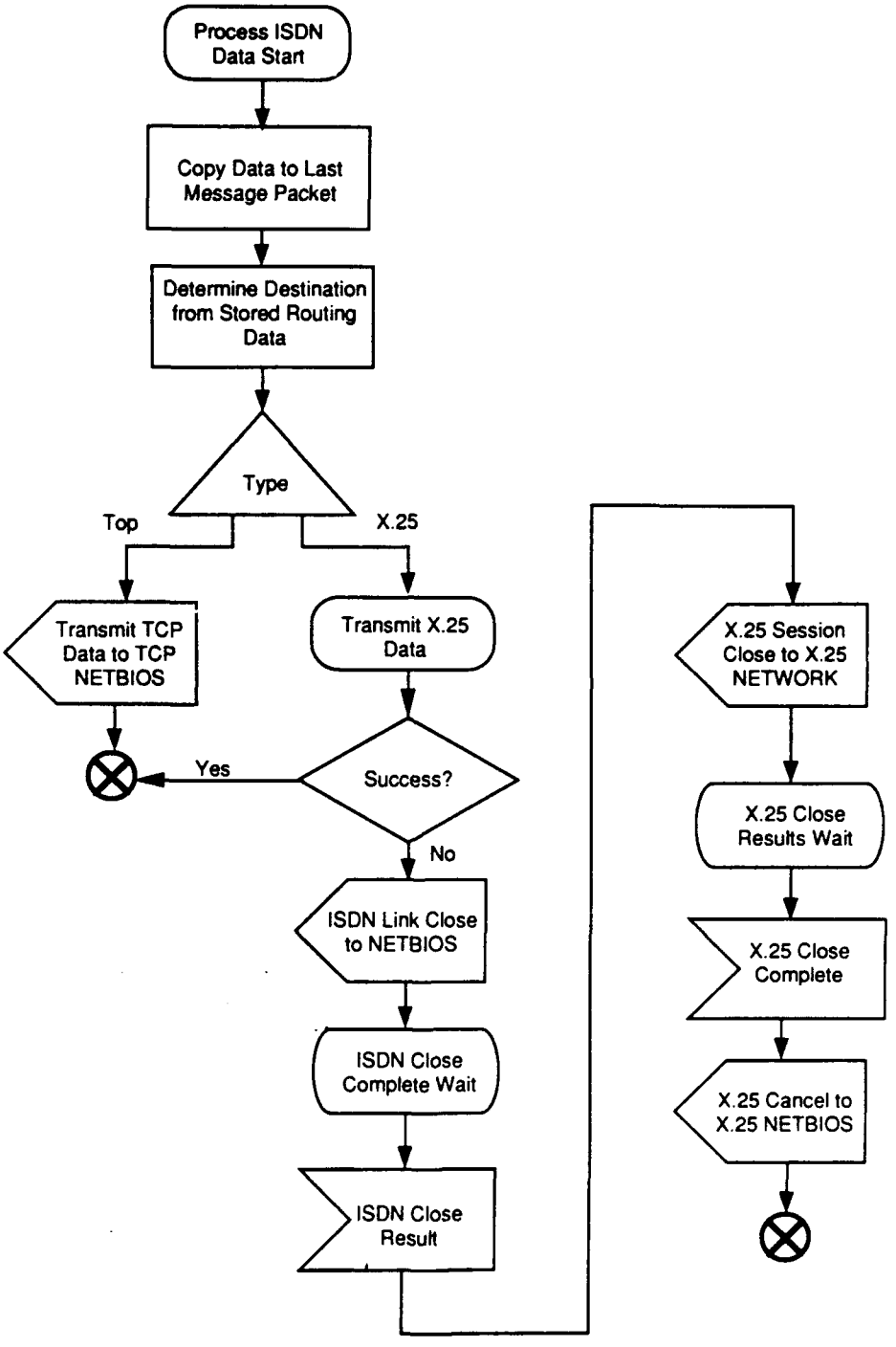


EXHIBIT 2-25

Process TCP Data

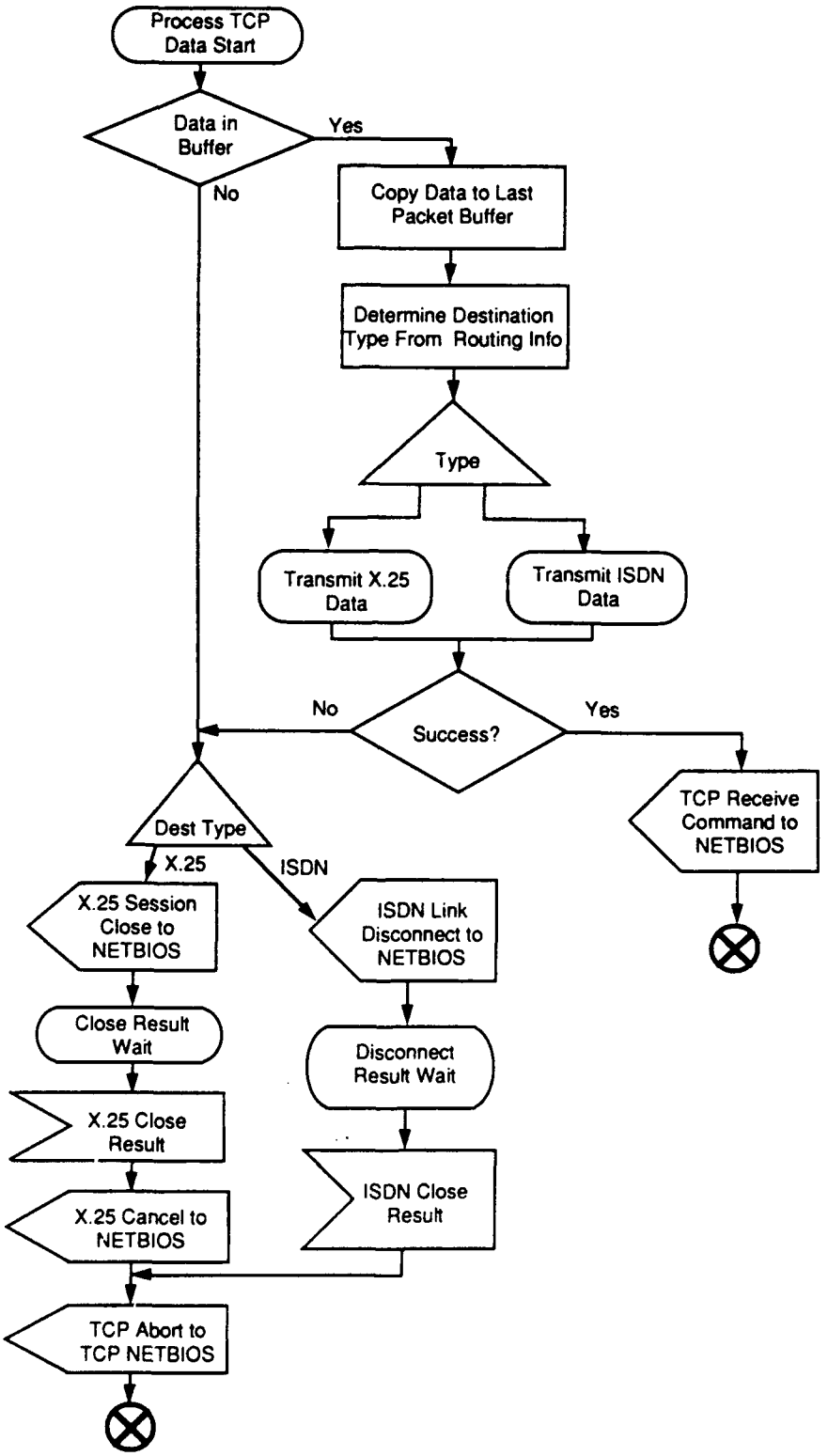


EXHIBIT 2-26

Process ISDN Header

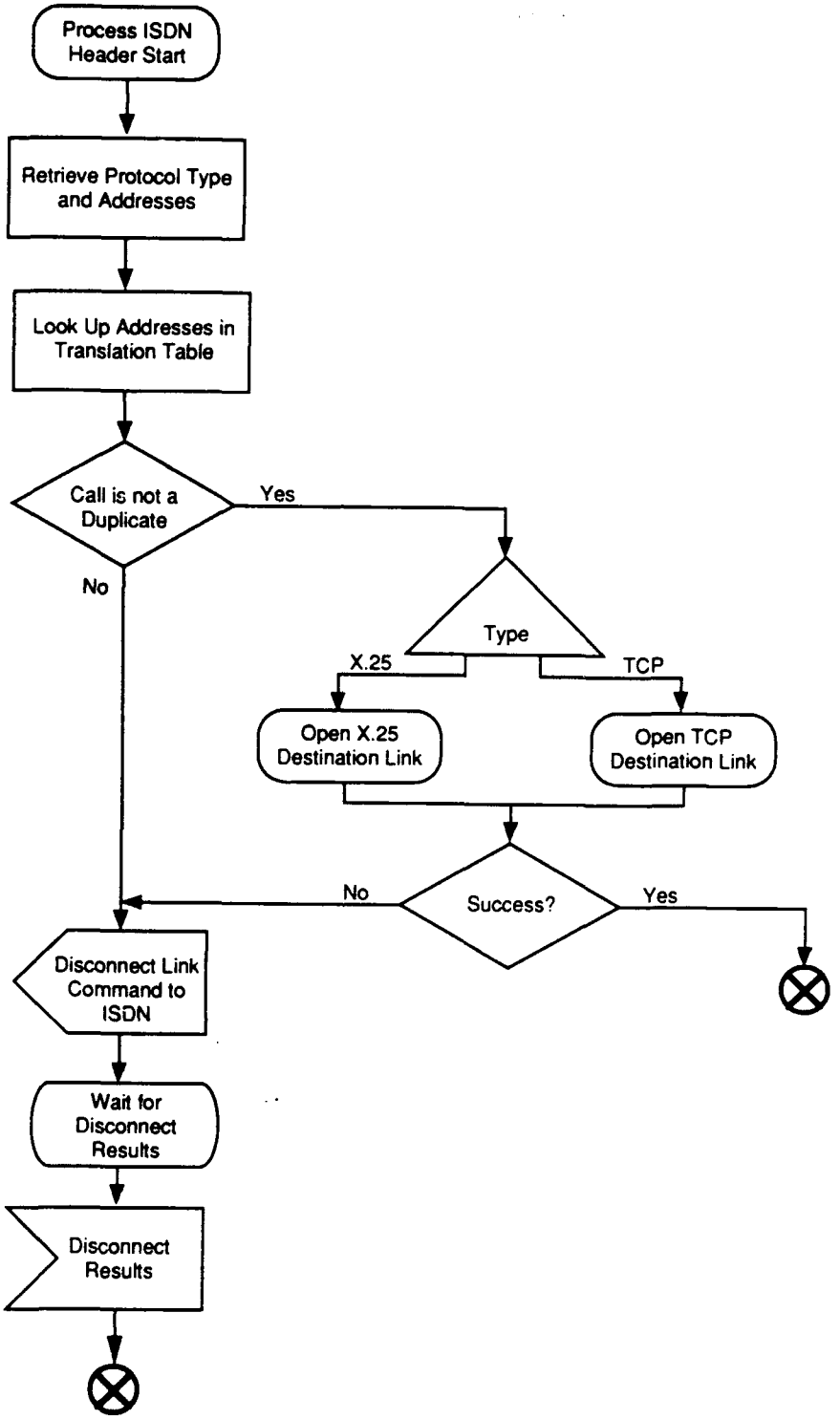


EXHIBIT 2-27

Process X.25 Remote Close

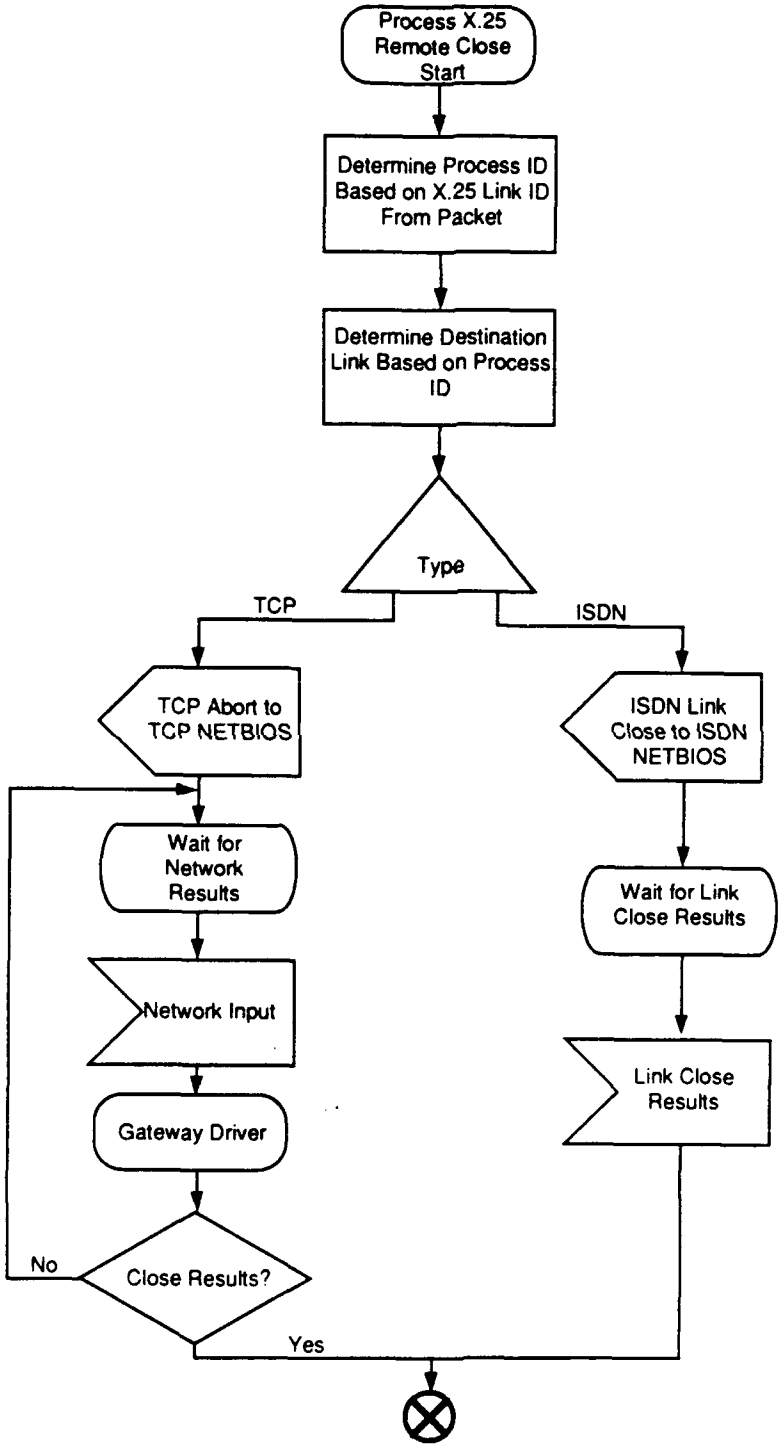


EXHIBIT 2-28

Process TCP Remote Close

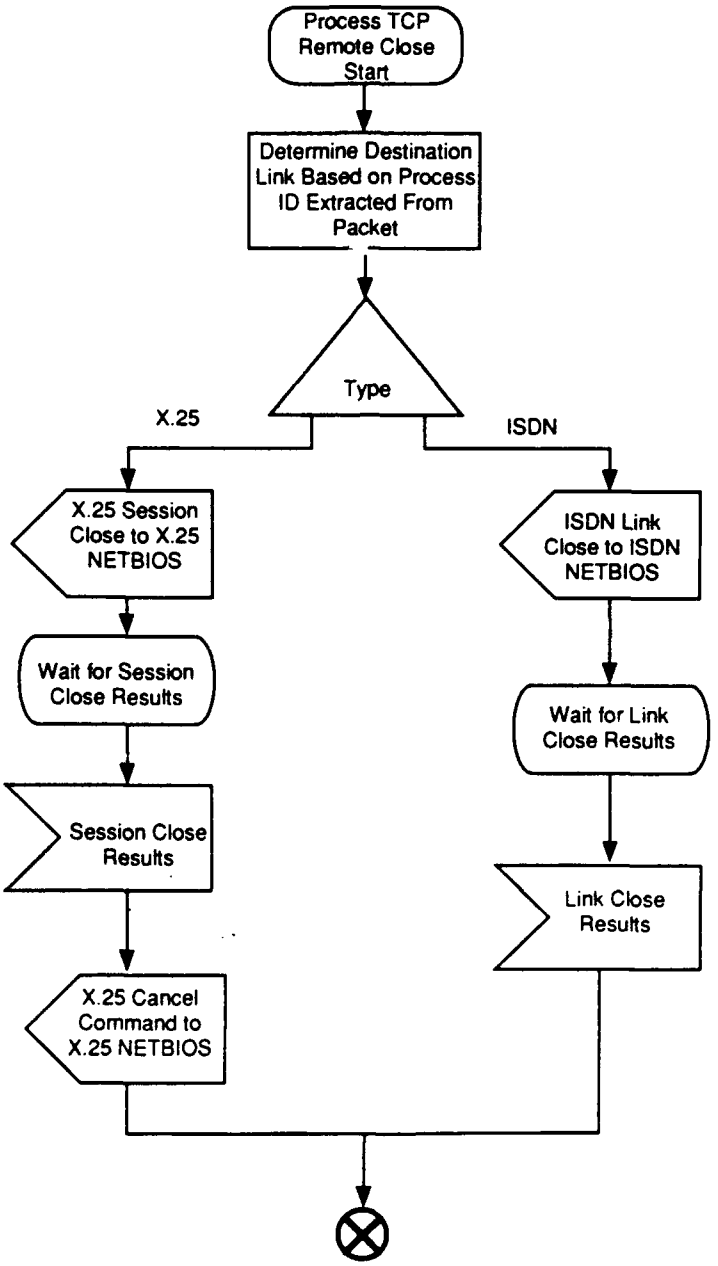


EXHIBIT 2-29

Process ISDN Remote Close

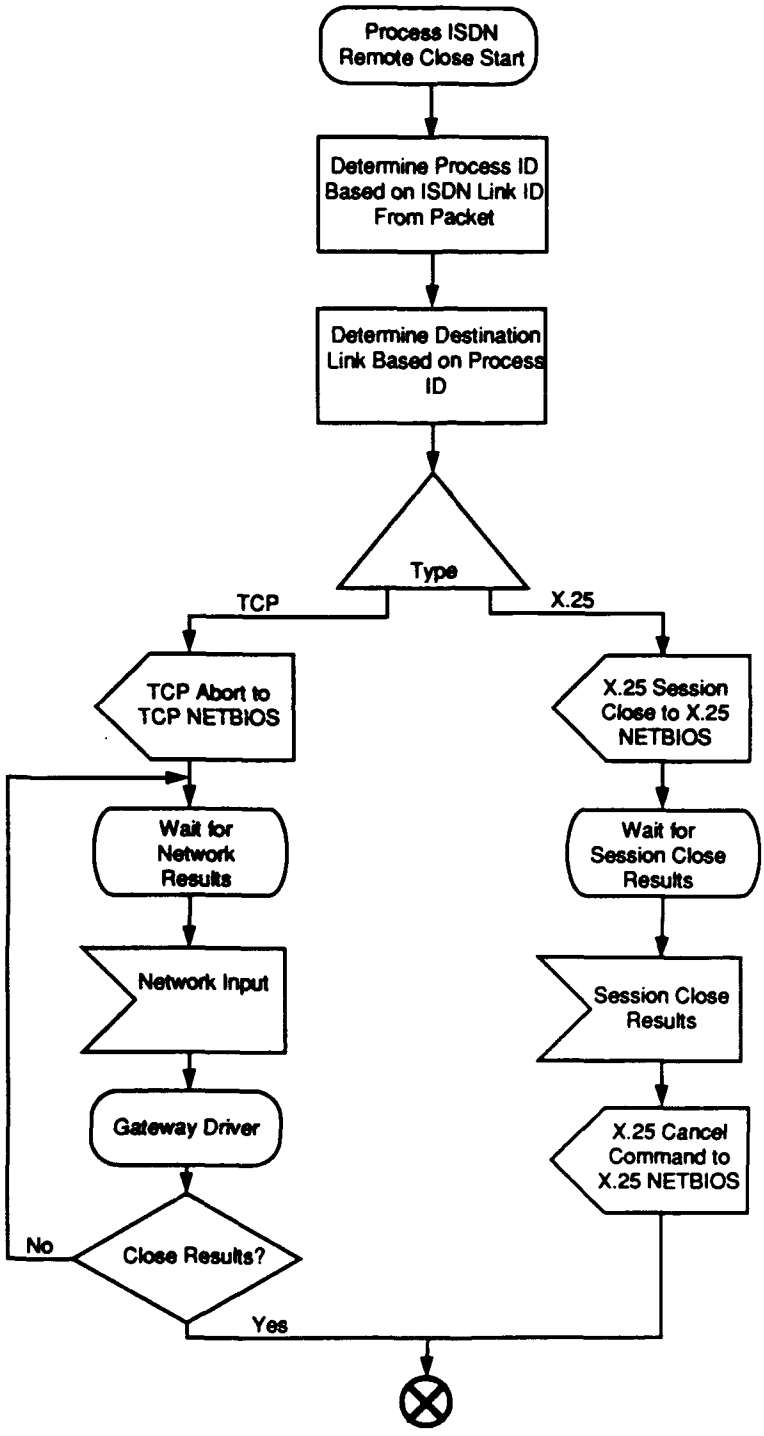


EXHIBIT 2-30

Process Transmit Conclusion

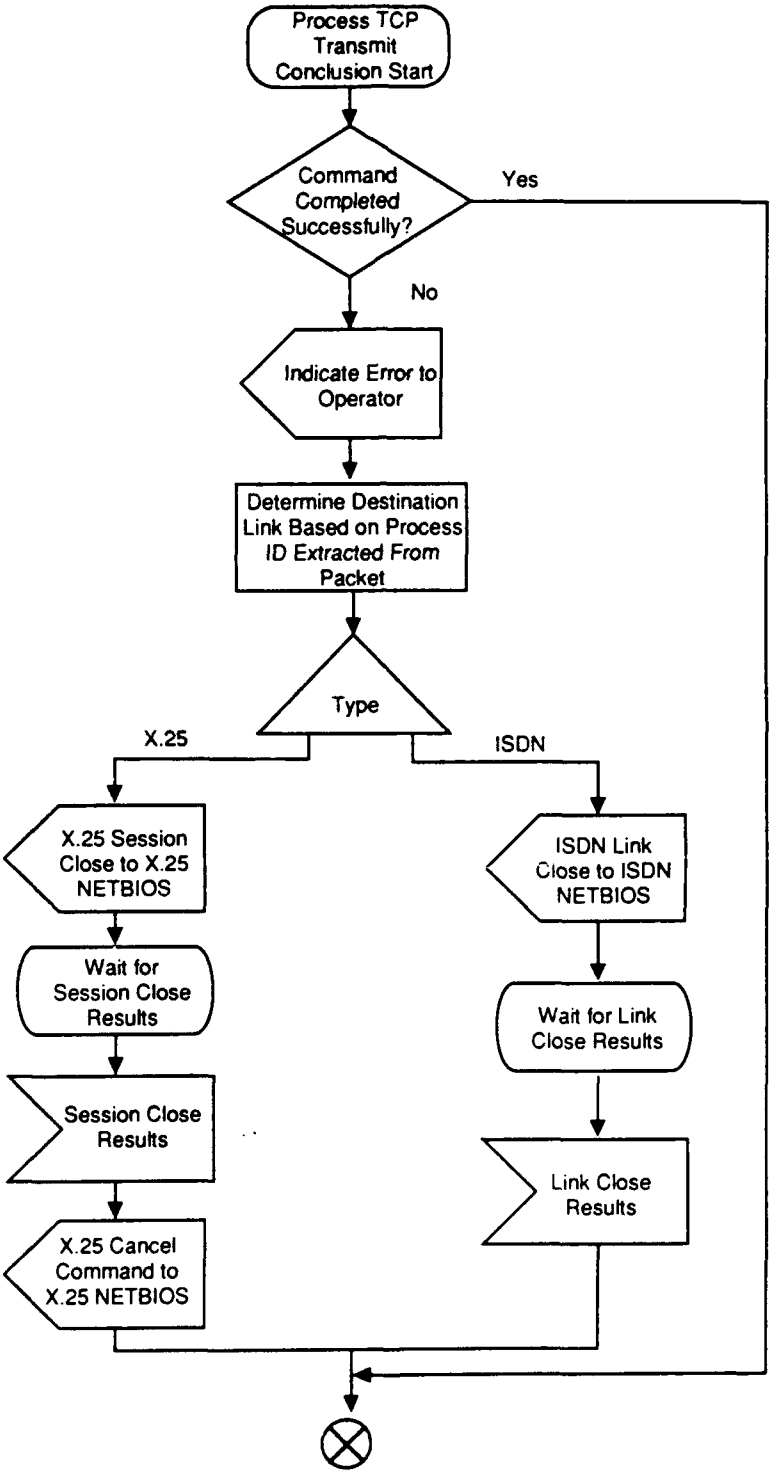


EXHIBIT 2-31

Open ISDN Link

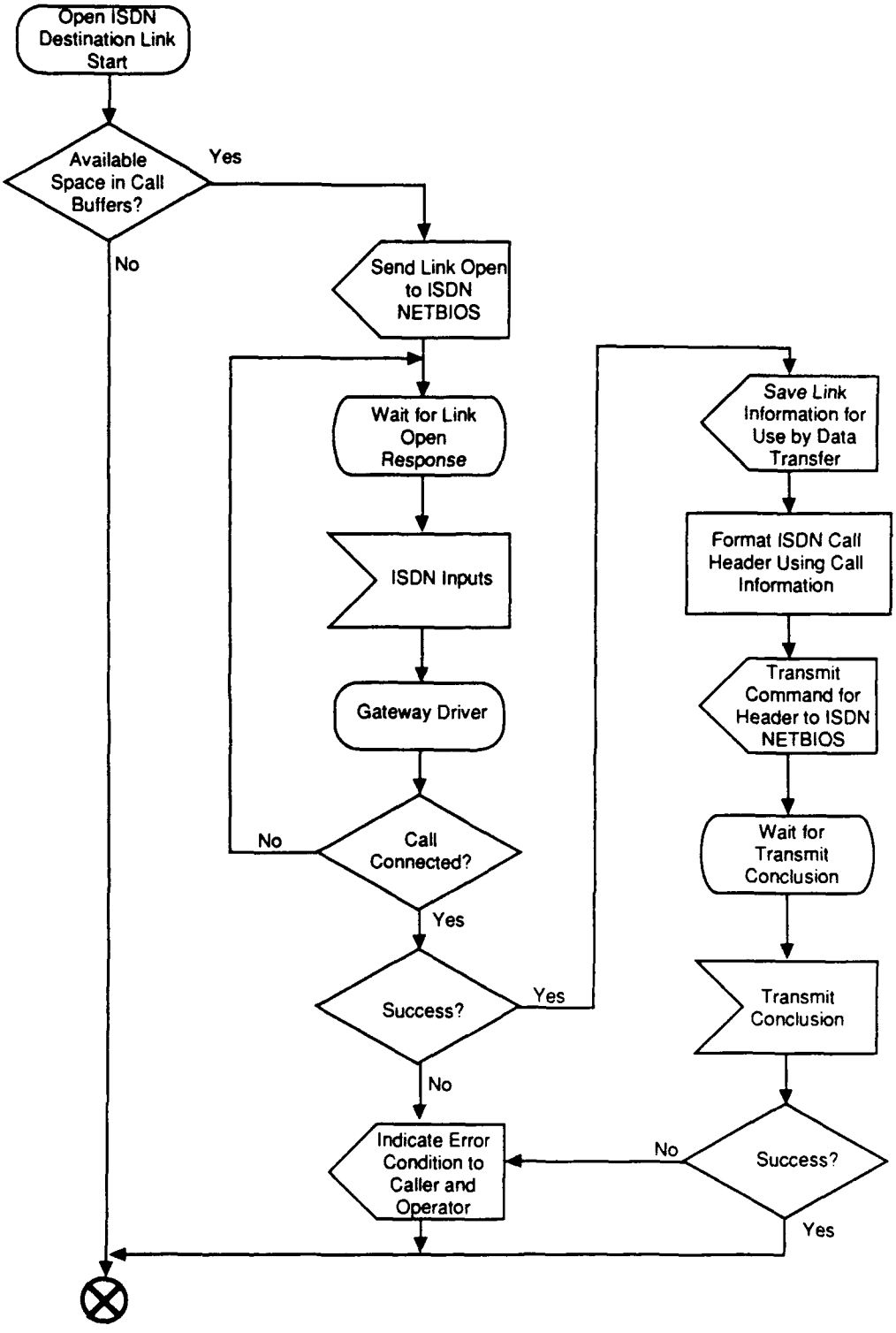


EXHIBIT 2-32

Open TCP Link

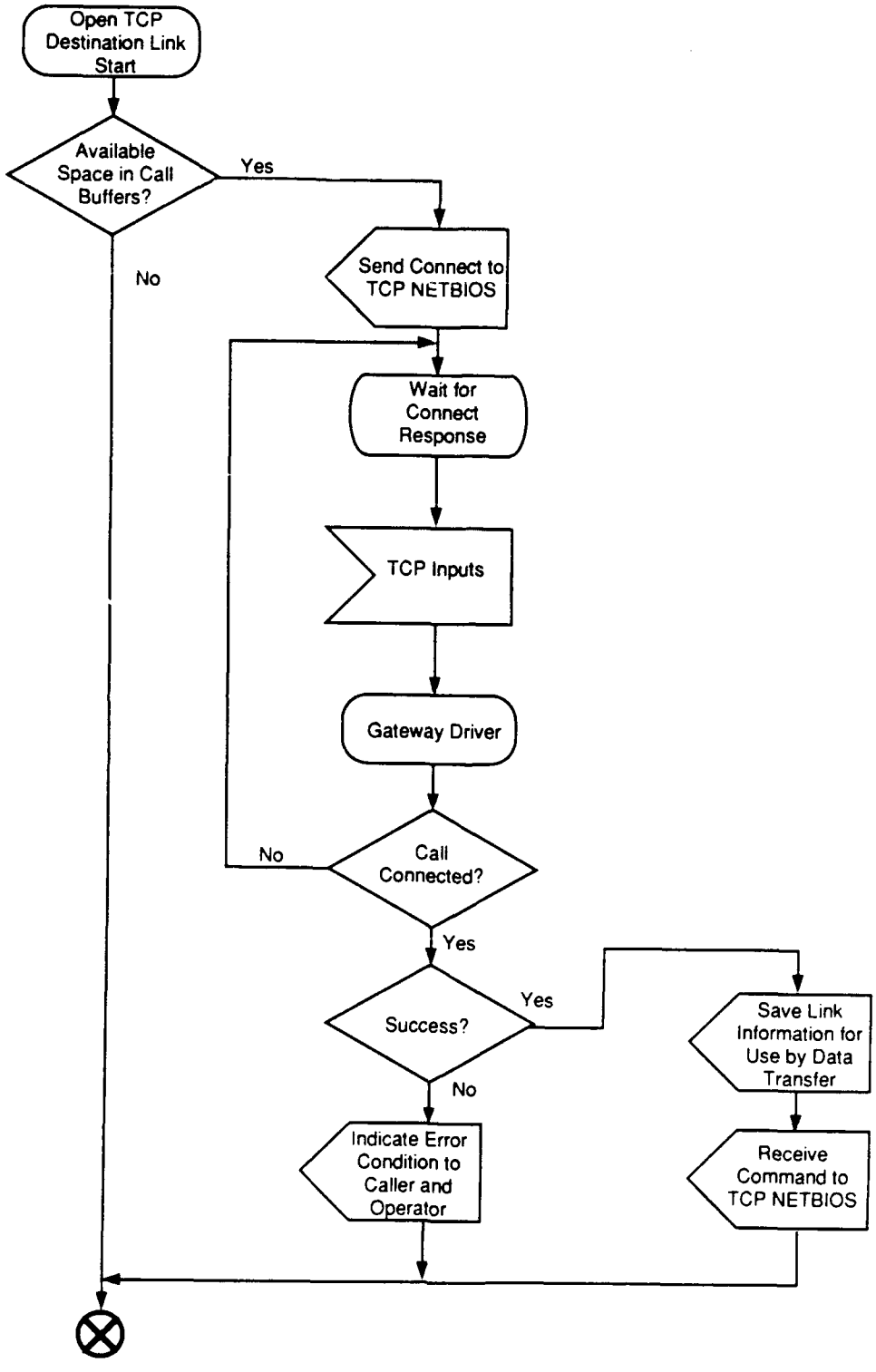


EXHIBIT 2-33

Open X.25 Link

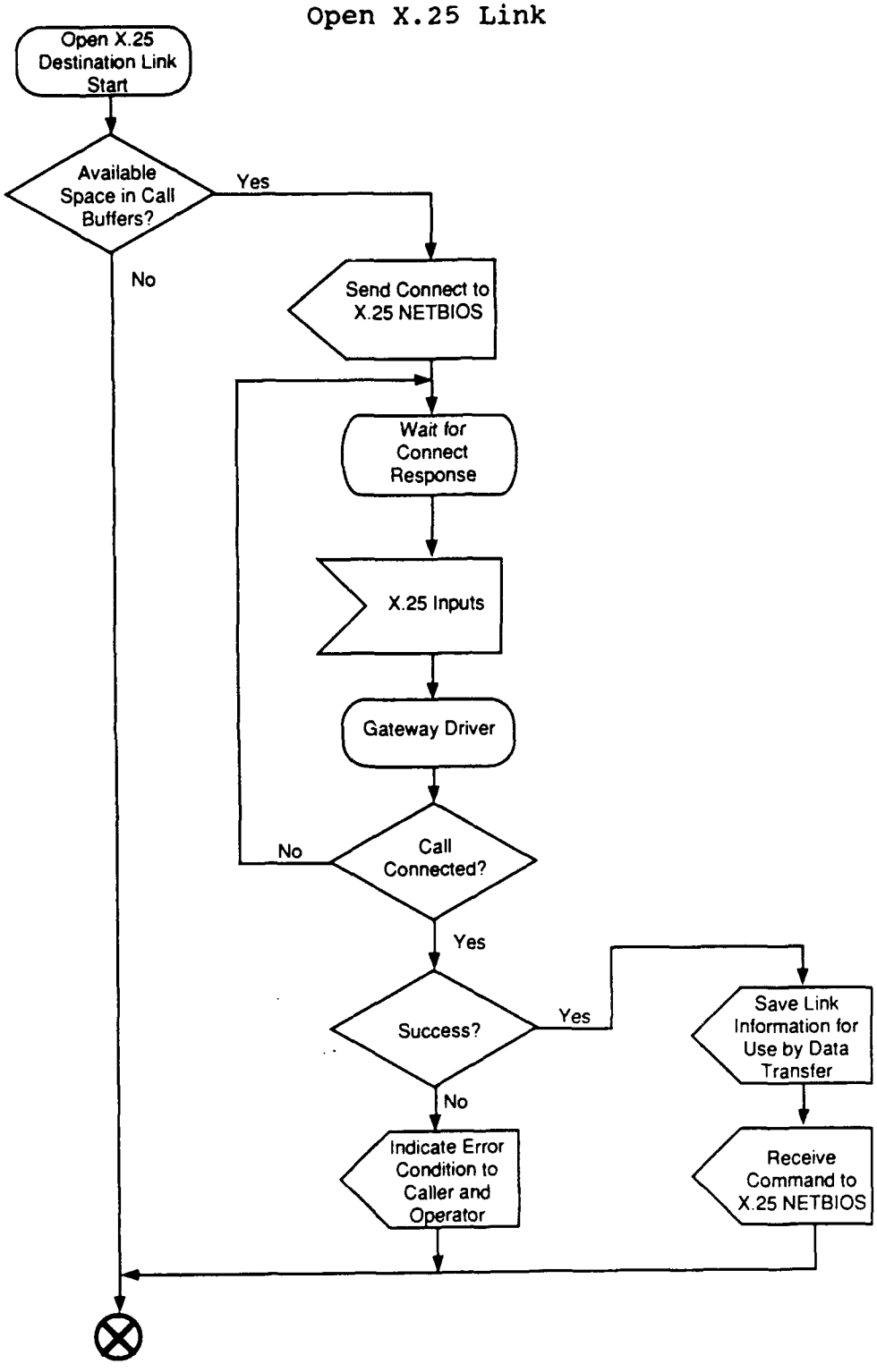


EXHIBIT 2-34

Transmit ISDN Data

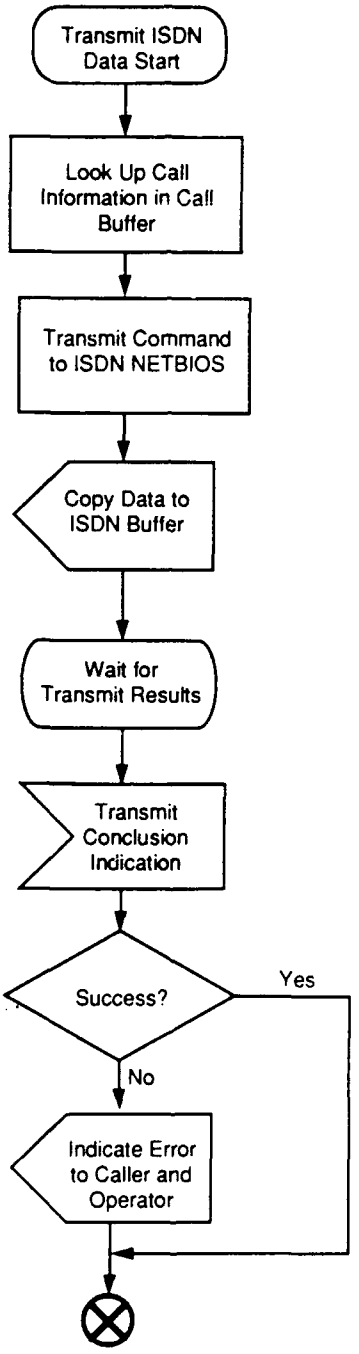
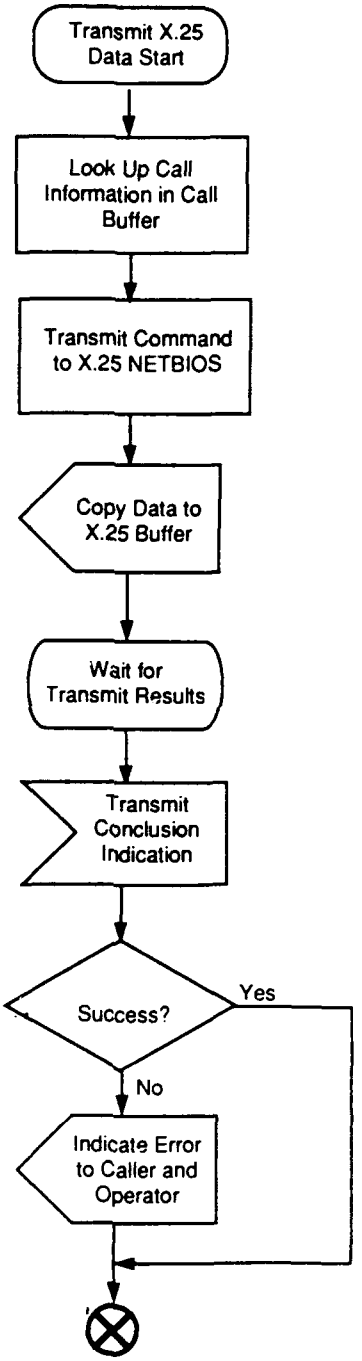


EXHIBIT 2-35

Transmit X.25 Data



L-88.35

3.0 GATEWAY SOFTWARE CODE

The code presented is the implementation of the detailed design shown in Section 2. The code was developed in the "C" programming language using Microsoft Quick C V1.1 and Microsoft Quick C Compiler for the DOS environment. The gateway code is compiled as 4 separate "C" modules. All other files are included with #include statements. The program is loaded using the "medium" memory model under the Microsoft "C" segmented memory mapping system.

Appendix A contains a list of files that are included on the gateway diskette. This diskette contains all of the necessary files to run the mutiprotocol gateway. What follows is a complete listing of the Booz, Allen developed gateway "C" code.

```
/*************************************************************/
/*** The following file contains code & include file    ***/
/*** statements for:                                    ***/
/***        1. Command Processing Task of MMI module  ***/
/***        2. Gateway Driver Module                   ***/
/***        3. Data Transfer module                    ***/
/***        4. Common Routines                         ***/
/***        5. Global Data                             ***/
/*** This file is Gatedev.c                            ***/
/*************************************************************/


#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <graph.h>
#include <math.h>
#include <timeb.h>

/* FRONTIER/TCP EQUATES AND COSTANTS */

#include "c:\tcp\include\ftctypes.h"
#include "c:\tcp\include\ulpuser.h"
#include "c:\tcp\include\ulpshare.h"

/* GATEWAY CONSTANTS */

#include "c:\devel\gatedef.h"

/*  GATEWAY TYPE DEFINITIONS */

#include "c:\devel\tipedef.h"


/*  GLOBAL DATA */

union
{
     struct data_message_type
     {
          short data_cmd;
          short data_LID;
          short data_Control;
          short data_Size;
          char data_Data[128];
     }dm;
     struct display_type
     {
          short dis_cmd;
```

```
          char dis_Message[128];
     }display;
     struct control_type

     {
          short ioctl_cmd;
          short ioctl_CODE;
          short ioctl_Size;
          char ioctl_Data[128];
     }ioctl;
}iu[MAX_X25_OUT+MAX_ISDN_OUT],ou[MAX_X25_OUT+MAX_ISDN_OUT],lpx,lp
i,
                              globdat,incall ;

     struct dpp_response
     {
          short link_op_res;
          short link_op_ret_code;
          short link_op_LID;
          short link_op_PCC;
          short link_op_cause;
          short link_op_diag;
          short link_op_session;
     }r;

/* Gateway translation/mapping table */

struct trans_table_type
  {
     char x25addr[X25_ADDR_LEN] ;  /* x.25 address field */
     ncb_t  *x25ptr             ;  /* x.25 listen pointer */
     char tcpaddr[TCP_ADDR_LEN] ;  /* TCP address field  */
     ushort tcpport              ; /* Port id to use */
     PCB *tcpptr                ;  /* TCP listen structure
pointer */
     char isdnaddr[ISDN_ADDR_LEN] ; /* ISDN address field */
  } trans_tbl[MAX_TBL] ;

STAT_ENTRY  ovr_stats,brk_stats[NUM_TYP] ;

/* The following defines the tables used for maintenance of
multiple */
/* connections at the same time */


OUT_CALL    x25_out[MAX_X25_OUT]
               ,tcp_out[MAX_TCP_OUT]
               ,isdn_out[MAX_ISDN_OUT] ;
```

```
LIOCTL  su ;


union  REGS regs;
struct SREGS sregs;

char            tracstrng[4][10] = {"[OFF]     ","[CONTROL]",
                                                "[DATA]    ","[BOTH]
 "} ;
char            numset[]="0123456789" ;
char            error_message [] = "This video mode is not
supported";
char            obuf[128];
char            buffer[255];
char            data_buffer[128] ;
char            tcpbuf[MAX_TCP_OUT][128] ;
char            tcplpr[128] ;
ushort          tcp_lpr_size = 0 ;
char            call_buffer[20] ;
char            perm_call_values[5]={0,1,0,0,0};
ushort          menu_choice[12]={0,0,0,0,0,0,0,0,0,0,0,0};
char            x25_send_address[X25_ADDR_LEN];
char            x25_recv_address[X25_ADDR_LEN];
char            isdn_send_address[ISDN_ADDR_LEN];
char            isdn_recv_address[ISDN_ADDR_LEN];
char            loc_isdn_addr[] = LISDNADDR ;
uchar           *srcp;
uchar far       *destp;
short           dpp_response = 0 ;
short           lid;
short           x25_call_sw;
short           isdn_call_sw;
uchar           x25_lsn;
uchar           sip_lsn;
uchar           x25d_lsn;
uchar           stats = 0;
uchar           gattrace = 0;
uchar           gateway_up = 0;
uchar           xlisten = 0;
short           hangup;
short           tcp_call_connected ;
short           tcp_stats_rx ;
short           isdn_call_connected ;
short           isdn_stats_rx ;
short           isdn_stats_retcode ;
ushort          inproc[MAX_X25_OUT] ;
long             tot_calls = 0 ;
long             tot_packs = 0 ;
long            call_queued_time = 0 ;
long            pack_queued_time = 0 ;
```

```
long            call_process_time = 0 ;
long            pack_process_time = 0 ;


/*  TCP status maintenance and interpretation variables */

short           tcp_call_count = 0;
short           tcp_packets_recv = 0;
ulong           tcp_bytes_recv = 0;
short           tcp_packets_sent = 0;
ulong           tcp_bytes_sent = 0;

char            tcp_stat_typ[11][10] = {
"Closed","Listen","Synrcvd","Synsent",

"Estab","Clswait","Finwait1",

"Finwait2","Closing","Timewait",
                                                "Lastack"} ;


long            starttime=0;
ushort          remote_length;
ncb_t           iincb[MAX_ISDN_OUT];          /* Input  ISDN NCB
*/
ncb_t           oincb[MAX_ISDN_OUT];          /* Output ISDN NCB
*/
ncb_t           ircvncb ;                    /*  wait ncb for isdn
*/
ncb_t           ixncb[MAX_X25_OUT];           /* Input  X25  NCB
*/
ncb_t           oxncb[MAX_X25_OUT];           /* Output X25  NCB
*/
UCMD            globucmd[MAX_TCP_OUT*2] ;     /* input & output
BCB'S */

/*  The following variables are used as a pending action queue */

ushort          frptr=0,bkptr=0 ;   /* Queue pointers */

struct  {   /* Queue definition */
ushort  typ ;   /* type of link rtequiring service */
ushort  act ;   /* action which just occurred */
ushort  indx ;  /* index into appropriate table (trans_tbl or
out_call) */
long    qtime ; /* time that the request was queued */
int  msec ;  /* milliseconds */
}   pend_act[MAX_PEND_ACT] ;

/* Added globals for TCP/X.25/ISDN modification */
```

```
char            x25text[] = "X.25 Address (DDDDDDDDDDDDDDD)"  ;
char            tcptext[] = "TCP  Address (DDD.DDD.DDD.DDD)"
  ;
char            isdntext[] = "ISDN Address (DDDDDDDDDDDDDDD)"  ;
char            porttext[] = "TCP Port Number (DD)" ;
char            menuchoice[] = "Current Menu Choices :" ;
char            menu1a[] = "A) Create New Entry. "       ;
char            menu1b[] = "B) Edit Current Address. "    ;
char            menu1c[] = "C) Edit Specified X.25 Address. " ;
char            menu1d[] = "D) Edit Specified TCP Address. " ;
char            menu1e[] = "E) Scroll to Next Address. " ;
char            menu1f[] = "F) Exit to Main Menu. " ;
char            menu2a[] = "Enter Current Address in Full." ;
char            menu2b[] = "<CR> to move to next item after
entry." ;
char            menu2c[] = "<DEL(CTRL-BKSP)> to delete entry." ;
char            menu2d[] = "<ESC> to exit screen." ;
char            menu2e[] = "Default ISDN address is local
address." ;
char            gatetitle[] = "X.25/TCP/ISDN Gateway v1.0" ;
char            mainmen[] = "Main Menu:\n" ;
char            editmen[] = "Address Translation Screen" ;
char            main1[] = "\t\t\t    Gateway Menu Options: \n\n"
;
ushort          entry_exists = 0 ;
int             sys_call_stat = 0 ;
char            prot_nam[2][4] = {"TCP","X25"} ;
ushort          cur_sess = 0 ;
ushort          firsttime = 1 ;

FILE            *logptr ;


#include "c:\devel\subdecl.h"


/*****************************************************************/
/***               COMMAND PROCESSING TASK MMI MODULE          ***/
/*****************************************************************/

main()
{
     uchar          ll;
     uchar          rl;
     ushort         i;
     char           ans;
     ushort         *initptr ;
     ncb_t far      *iinp;
     ncb_t far      *oinp;
```

```
ncb_t far       *ixnp;
ncb_t far       *oxnp;
ncb_t far       *lxnp;   /* Listen X25 ncb */

i = 0;

initptr = (short *) &ovr_stats ;
for( i = 0; i < sizeof(ovr_stats) ; i++)
{
     *initptr++ = 0 ;
}
initptr = (short *) &brk_stats[0] ;
for( i = 0; i < (3*sizeof(ovr_stats)) ; i++)
{
     *initptr++ = 0 ;
}


logptr = fopen("GATLOG.LOG","w") ;

main_menu();
hangup = 0;
ans = 0;
x25_call_sw = 0;
isdn_call_sw = 0;
entry_exists = READ_TRANS_TBL() ;



while(ans!='L')
{
          ans = WAIT_USER_INPUT() ;
          ans = toupper(ans) ;
          if(menu_choice[(ans-65)]==1)
          {
               sprintf(obuf, "Menu choice already
selected.\n");
               output_message(obuf);
          }
          else
          {
          switch(ans)
          {
          case 'A' :
               if (entry_exists == 1)
               {

                    if (gattrace == 1)
                    {
                         sys_call_stat =
```

```
system("X25_TRACE");
                              }
                              else
                              {
                                   _settextposition (23,7);
                                   sys_call_stat =
system("X25_START");
                              }
                              sprintf(obuf,
                                   "X.25 Load Completed, Status of
system call : %d\n",
                                                        sys_call_stat);
                              output_message(obuf) ;
                              menu_choice[0] = 1;
                         }
                         break;
                    case 'B' :
                         if (entry_exists == 1)
                         {
                              if (gattrace == 1)
                              {
                                   _settextposition (24,1);
                                   sys_call_stat =
system("ISDN_TRACE");
                              }
                              else
                              {
                                   _settextposition (23,7);
                                   sys_call_stat =
system("ISDN_START");
                              }

                              sprintf(obuf,
                                   "ISDN Load Completed, Status of
system call : %d\n",

sys_call_stat);
                              output_message(obuf) ;
                              menu_choice[1] = 1;
                         }
                         break;
                    case 'C' :
                         INIT_TCP_LINK() ;
                         menu_choice[2] = 1 ;
                         break ;
                    case 'D' :

SETUP_GATE(lxnp,menu_choice[0],menu_choice[1],menu_choice[2]) ;
                         gateway_up = 1 ;
                         break;
```

```
                    case 'E' :
                         _settextposition(12,57);
                         if (++gattrace == 4) gattrace = 0 ;
                         printf("%s\n",tracstrng[gattrace]) ;
                         break;
                    case 'F' :
                         shutdown();
                         for (i=0;i<8;i++)
                              menu_choice[i] = 0;
                         gattrace = 0;
                         break;
                    case 'G':
                         trans_stats();
                         break;
                    case 'H' :
                         x25_stats();
                         break;
                    case 'I' :
                         isdn_stats();
                         break;
                    case 'J' :          /* Display TCP statisitics */
                         TCP_STATS() ;
                         break ;
                    case 'K' :          /* Edit translation table */
                         ED_TRANS_TBL() ;
                         break ;
                    case 'L':           /* Exit to DOS */
                         _setvideomode(_DEFAULTMODE);
                         break;
                    default :
                         sprintf(obuf,"Please stick to reality...\n");
                         output_message(obuf);
                    }
                    }
                    if (ans < 'L'&& ans > 'C' && ans != 'E')
main_menu();

               CHK_NET() ;
               selection();
          }
          fclose(logptr) ;
          STORE_TRANS_TBL(entry_exists) ;
/*
          End.
*/
}


long collarr[16] =
{_BLACK,_BLUE,_GREEN,_CYAN,_RED,_MAGENTA,_BROWN,
```

```
_WHITE,_GRAY,_LIGHTBLUE,_LIGHTGREEN,_LIGHTCYAN,
                                _LIGHTRED
,_LIGHTMAGENTA,_LIGHTYELLOW,_BRIGHTWHITE} ;

void main_menu()
/*
     Main Menu Screen routine.
*/
{
     struct videoconfig vc;



/*       SET BACKGROUND
*/
     _setvideomode(_ERESCOLOR) ;
     _remapallpalette(&collarr) ;
     _setbkcolor(_BLUE);
     _clearscreen(_GCLEARSCREEN) ;
     _getvideoconfig (&vc);
     _setlogorg (vc.numxpixels/2-1,0);
     _setcolor(9);
     _rectangle(_GBORDER, -299,1,299,39);


/*       ADD TEXT
*/
     _settextcolor (10);
     _settextposition (2,27);
     _outtext (gatetitle);
     _settextcolor (15);
     _settextposition (5,37);
     _outtext (mainmen);
     printf(main1) ;
     printf("\t\t\t    A)  Reset and Load X.25 \n");
     printf("\t\t\t    B)  Reset and Load ISDN \n");
     printf("\t\t\t    C)  Reset and Load TCP \n") ;
     printf("\t\t\t    D)  Execute Gateway Startup ");
     _settextposition(11,57);
     if(gateway_up==0)
          printf("[OFF]\n");
     else
          printf("[ON] \n");
     printf("\t\t\t    E)  Toggle Gateway Trace    %s \n
",tracstrng[gattrace]);
     printf("\t\t\t    F)  Clear Calls\n");
     printf("\t\t\t    G)  View Transmission Statistics\n");
     printf("\t\t\t    H)  View X25  Protocol Statistics\n");
     printf("\t\t\t    I)  View ISDN Protocol Statistics\n");
```

```
     printf("\t\t\t    J)  View TCP Protocol Statistics\n");
     printf("\t\t\t    K)  Edit Address Translation Table \n");
     printf("\t\t\t    L)  Exit to DOS\n\n");


     _settextcolor (3);
     _settextposition (20,28);
     sprintf(buffer, " Please enter a selection:  ");
     _outtext (buffer);
     selection();

/*      ADD TOP RECTANGLE
*/
     _setcolor(15);
     _setlinestyle(0xFFFF);
     _rectangle(_GBORDER, -300,0,300,40);
     _rectangle(_GBORDER, -300,44,300,296);
     _rectangle(_GBORDER, -299,45,299,295);
/*      ADD LOWER RECTANGLE
*/
     _setcolor(7);
     _setlinestyle(0xFFFF);
     _rectangle(_GBORDER, -300,300,300,340);
     _rectangle(_GBORDER, -299,301,299,339);
/* } */
}


/*****************************************************************/
/***                      COMMON ROUTINES                      ***/
/*****************************************************************/


void selection()
{
/*  This routine blanks the command input slot and repositions
cursor */

     _settextposition(20,56);
     printf("   ");
     _settextposition(20,56);
}

void output_message(char *bufptr)
{
/*  This routine writes a message to the message window and to
the log */
/*      if logging is enabled */

     char blanks[71];
```

```
     int     i, init=0;

     if(init==0)
     {
          for(i=0; i<71; i++)
               blanks[i] = ' ';
          blanks[71] = 0x0;
          init = 1;
     }
     _settextposition (23,7);
     _outtext (blanks);
     _settextposition (23,7);
     _outtext (bufptr);
     if (gattrace != 0)
     {
          fprintf(logptr,"%s\n",bufptr) ;
          fflush(logptr) ;
     }
}

/*
     Subroutine to acutally send out the command using the
     60 interrupt vector.
*/
void send_cmd_60(ncb_t far *ncbp)
{
     short ret      = 0;

     regs.x.bx      = FP_OFF(ncbp);
     sregs.es       = FP_SEG(ncbp);
     ret            = int86x(NETBIOS_60,&regs,&regs,&sregs);

     if((ncbp->ncb_retcode>0)&&(ncbp->ncb_retcode!=0xff))
     {
          sprintf(obuf,"ISDN : int86x retcode: 0x%x NCB retcode
: 0x%x ",
               ret&0x00ff,ncbp->ncb_retcode);
          output_message(obuf) ;
     }

}
/*
     Subroutine to clear out the NCB.
*/
void clear_ncb(ncb_t far *ncbp)
{
     int i;
     char far *cptr = (char far *) ncbp;
     for (i=0; i<sizeof(*ncbp); i++,cptr++)
          *cptr = '\x00';
```

```
}
void interrupt far post_rtn()
{
     dpp_response = 1 ;
}
/*
     5C interrupt vector.
*/
void send_cmd_5c(ncb_t far *ncbp)
{
     short ret      = 0;
     int i;
     ncb_t far *lp;
     regs.x.bx      = FP_OFF(ncbp);
     sregs.es       = FP_SEG(ncbp);
     lp = ncbp;
     ret            = int86x(NETBIOS_5C,&regs,&regs,&sregs);

     if((ncbp->ncb_retcode>0)&&(ncbp->ncb_retcode!=0xff)
          &&(ncbp->ncb_retcode!=0xa))
     {
          sprintf(obuf,"X25   : int86x ret: 0x%x NCB ret    : 0x%x
",
               ret&0x00ff,ncbp->ncb_retcode);
          output_message(obuf) ;
     }

}

void interrupt far x25_listen_post()
/*
     Post routine to listen for X25 call.
*/
{
     xlisten = 1 ;
}

void interrupt far isdn_send()
/*
     ISR for ISDN X25 D channel send.
*/
{
     isdn_call_sw = 1;
}
void interrupt far x25_send()
/*
     ISR for X25 send.
*/
{
```

```
     x25_call_sw = 1 ;
}
/*****************************************/
/*** The following routines are routines ***/
/*** provided by Frontier to interface   ***/
/*** with the ADCOM 2i board.            ***/
/*****************************************/

#include <c:\tcp\libulp\neterr.c>
#include <c:\tcp\libulp\tcpctl.c>
#include <c:\tcp\libulp\ulprxcmd.c>
#include <c:\tcp\libulp\tcprx.c>
#include <c:\tcp\libulp\tcptx.c>

/*****************************************/

/*
This routine waits for user input and checks for network activity
*/
#include <c:\devel\wa_us_in.c>

/*
This routine performs the common ISDN NCB actions
*/
#include <c:\devel\com_is_act.c>

/*****************************************************************/
/***                  GATEWAY DRIVER MODULE                    ***/
/*****************************************************************/


void QUE_X25_LISN()
{

/*  This routine queues X.25 listen indications */


     int ii ;
     struct timeb timebuffer ;

/*  loop for all translation table entries */
     for(ii=0;ii<MAX_TBL ; ii++)
     {

/*      If the current X.25 entry has a listen pending */
/*      Then */
        if (trans_tbl[ii].x25ptr != 0)
        {

/*        If listen is completed */
```

```
/*          Then */
               if (trans_tbl[ii].x25ptr->ncb_retcode == 0)
               {

/*              Queue the result */
                    ftime(&timebuffer) ;
                    pend_act[frptr].qtime = timebuffer.time ;
                    pend_act[frptr].msec = timebuffer.millitm ;
                    pend_act[frptr].typ = X25TYP ;
                    pend_act[frptr].act = OP_LISTEN ;
                    pend_act[frptr].indx = ii ;
                    trans_tbl[ii].x25ptr->ncb_retcode = 99 ;
                    if (++frptr == MAX_PEND_ACT)
                    {
                         frptr = 0 ;
                    }
               }
          }
     }



}

void QUE_X25_RX()
{
/*  This routine queues X.25 receives */

     int ii ;
     struct timeb timebuffer ;


/*  Loop through the X.25 receive buffers */
     for(ii = 0; ii < MAX_X25_OUT ; ii++)
     {

/*      if current receive process is complete and not in
progress */
/*      then */

          if (ixncb[ii].ncb_retcode != 0xff && inproc[ii] !=
TRUE)
          {
/*          fill out the pending action queue */

               ftime(&timebuffer) ;
               pend_act[frptr].qtime = timebuffer.time ;
               pend_act[frptr].msec = timebuffer.millitm ;
               pend_act[frptr].indx = ii ;
               pend_act[frptr].typ = X25TYP ;
```

```
                    pend_act[frptr].act = OP_RX ;
                    inproc[ii] = TRUE ;
                    if (++frptr == MAX_PEND_ACT)
                    {
                         frptr = 0 ;
                    }
               }
          }

}

void x25_call_wait(ushort row_use)
/*
     Setup Wait routine to receive data from the remote X25 site.
*/
{
     short i;
     ncb_t far* ixnp ;

     ixnp = &ixncb[row_use] ;
     rtn_ptr = x25_send;
     clear_ncb(ixnp);
     srcp = X25_NAME;
     destp = ixnp->ncb_callname;
     for(i=0;i<NB_NAME_LEN;i++,srcp++,destp++)
          *destp = *srcp;
     ixnp->ncb_command = 0x95;
     ixnp->ncb_lsn = x25_out[row_use].cur_typ_id ;
     ixnp->ncb_lana_num = 0x0ff;
     ixnp->ncb_buffer = iu[row_use].dm.data_Data;
     ixnp->ncb_length = 128;

     ixnp->ui2 = FP_SEG(rtn_ptr);
     ixnp->ui1 = FP_OFF(rtn_ptr);

     inproc[row_use] = FALSE ;

     send_cmd_5c(ixnp);
}

int isdn_call_wait()
/*
     Setup Wait routine to receive data from the ISDN X25D site.
*/
{
     short i;
     ncb_t *iinp ;

     rtn_ptr = isdn_send;
     iinp = &ircvncb ;
```

```
     PERF_COM_NCB_ACT(&ircvncb) ;
     iinp->ncb_buffer = (char *) &globdat.dm;
     iinp->ncb_length = sizeof(globdat.dm);
     iinp->ncb_command = 0x95;

     iinp->ui2 = FP_SEG(rtn_ptr);
     iinp->ui1 = FP_OFF(rtn_ptr);

     isdn_call_sw = 0 ;
     send_cmd_60(iinp);

     if (iinp->ncb_retcode == 0 || iinp->ncb_retcode == 0xff)
          return(1) ;
     else

     {
          sprintf(obuf,"ISDN receive queue error
%d",iinp->ncb_retcode) ;
          output_message(obuf);
          return(0) ;
     }
}

/*
This routine provides the main gateway interface with the driver
module
*/
#include <c:\devel\chk_net.c>

/*
This routine determines which X.25 action to perform as a result
of the
queued input.
*/
#include <c:\devel\det_x_act.c>

/*
This routine determines which TCP action to perform as a result
of the
queued input.
*/
#include <c:\devel\det_t_act.c>

/*
This routine determines which ISDN action to perform as a result
of the
queued input.
*/
#include <c:\devel\det_i_act.c>
```

```
/*
This routine queues TCP messages received from the ADCOM 2i
NETBIOS
*/
#include <c:\devel\q_tc_act.c>

/*
These routines queue ISDN messages received from the TELIOS
NETBIOS
*/
#include <c:\devel\q_is_act.c>
#include <c:\devel\q_is_con.c>
#include <c:\devel\q_is_rcv.c>

/*****************************************************************/
/***                    DATA TRANSFER MODULE                   ***/
/*****************************************************************/


void isdn_call_setup(ushort lid,ncb_t far *oinp,ushort row_use)

/*
     Setup the output to ISDN X25D passing structure.
*/
{
     short i;

     PERF_COM_NCB_ACT(oinp) ;
     oinp->ncb_command = 0x14;
     oinp->ncb_buffer = (char *) &ou[row_use].dm;
     oinp->ncb_length = sizeof(ou[row_use].dm);
     ou[row_use].dm.data_cmd = 0x01;
     ou[row_use].dm.data_LID = lid;
     ou[row_use].dm.data_Control = 0x0;
}

void x25_call_setup(ushort row_use)
/*
     Setup the output to X25 passing structure.
*/
{
     short i;
     ncb_t   far *oxnp ;


     oxnp = &oxncb[row_use] ;
     clear_ncb(oxnp);
     destp = oxnp->ncb_callname;
     srcp = X25_NAME;
     for(i=0;i<NB_NAME_LEN;i++,srcp++,destp++)
```

```
            *destp = *srcp;
        oxnp->ncb_command = 0x14;
        oxnp->ncb_lsn = x25_out[row_use].cur_typ_id ;
        oxnp->ncb_buffer = ou[row_use].dm.data_Data;
        oxnp->ncb_length = sizeof(ou[row_use].dm.data_Data) ;
        oxnp->ncb_lana_num = 0x0ff;
}

/*
This routine processes received X.25 data packets
*/
#include <c:\devel\prx_x_dat.c>

/*
This routine processes received TCP data packets
*/
#include <c:\devel\prx_t_dat.c>

/*
This routine processes received ISDN data packets
*/
#include <c:\devel\prx_i_dat.c>



/*
This routine initiates a data transmission to the link type
determined by the call routing task of the Call Processing module
*/
#include <c:\devel\in_tx_ds.c>

/*
This routine transmits a data packet on the TCP link
*/
#include <c:\devel\tx_tc_msg.c>

/*
This routine transmits a data packet on the ISDN link
*/
#include <c:\devel\tx_is_msg.c>

/*
This routine transmits a data packet on the X.25 link
*/
#include <c:\devel\tx_x2_msg.c>

/*
This routine processes Transmit concluded messages from
the ADCOM 2i NETBIOS
*/
```

```
#include <c:\devel\ptx_tc_con.c>
#include <c:\devel\stubs.c>
```

```
char    addr_err1[] = "Cannot start TCP link without translation
entry" ;

void    INIT_TCP_LINK(ushort ent_exist)
{
     int sys_call_stat ;

/*  This routine initiates the dos command file which starts the
TCP link */

/*  If at least 1 entry exists in the translation table */
/*  Then */
/*      Initiate the system call to initialize the Adcom */

     if (ent_exist != 0)
     {
          if (gattrace == 1 || gattrace == 3)
          {
               _settextposition (24,1);
               sys_call_stat = system("TCP_TRACE");
          }
          else
          {
               _settextposition (23,7);
               sys_call_stat = system("TCP_START");
          }

          sprintf(obuf,"TCP Load Completed, Status of system call
: %d\n",
                                                  sys_call_stat);
          output_message(obuf) ;
     }
     else
     {
          output_message(addr_err1) ;
     }
}
```

```
char WAIT_USER_INPUT()
{
     char    inpkey ;

/* This routine gets 1 character of user input, checking for
Network  */
/* activity while waiting   */

/*  Loop until some input received */

     inpkey = 0 ;

     while ( inpkey == 0 )
     {

/*       Check for network activity  */

           CHK_NET() ;

/*       If the keyboard has been hit since the last check   */
/*       Then   */
/*           Get the input key   */

           if (kbhit() != 0)
           {
                inpkey = getch() ;
           }
     }

     return(inpkey) ;
}
```

```
void  ED_TRANS_TBL()
{

/*  This routine controls the user edit of the protocol
translation table */

     char    edans ;
     ushort  currow ;

/*  Display the Edit screen */

     DISP_SCRN(0,1,1,1) ;

/*  Loop until the exit command is encountered  */

     currow = 0 ;
     edans = 0 ;

     while (edans != 'F')
     {

/*      Wait for user input */

          edans = WAIT_USER_INPUT() ;
          edans = toupper(edans) ;

/*      determine and perform the appropriate action */

          currow = DET_ED_ACT(edans,currow) ;

     }
}
```

```
char    blnkbuf[] = "                                                  " ;

void    DISP_SCRN(ushort disprow, ushort menuid, ushort backflg,
                                ushort valflg)
{

/* This routine displays the requested prtions of the edit screen
*/
/*  Available subportions are background,menu or values */

     char specchar ;


/*  If the background is to be displayed */
/*  Then */

     if (backflg == 1)
     {

/*      Set background  */
/*      Draw background rectangles */

          _setbkcolor(_BLUE) ;
          _clearscreen(_GCLEARSCREEN) ;
          _setcolor(15) ;
          _setlinestyle(0xFFFF) ;

/*      Title rectangle */

          _rectangle(_GBORDER,-300,0,300,40) ;

/*      menu rectangle */

          _rectangle(_GBORDER,-300,300,300,340) ;

/*      main rectangle */

          _rectangle(_GBORDER,-300,44,300,296) ;

/*      Title rectangle */

          _setcolor(9) ;
          _rectangle(_GBORDER,-299,1,299,39) ;

/*      Title rectangle */

          _rectangle(_GBORDER,-299,301,299,339) ;

/*      Title rectangle */
```

```
          _rectangle(_GBORDER,-299,45,299,295) ;


/*       Place in fixed text */

          _settextcolor(10) ;
          _settextposition(2,27) ;
          _outtext(gatetitle) ;
          _settextcolor(15) ;
          _settextposition(6,27) ;
          _outtext(editmen) ;
          _settextposition(8,10) ;
          _outtext(x25text) ;
          _settextposition(10,10) ;
          _outtext(tcptext) ;
          _settextposition(11,10) ;
          _outtext(porttext) ;
          _settextposition(13,10) ;
          _outtext(isdntext) ;
     }

/*  If menu is indicated to be diplayed */
/*  Then */
/*       Display indicated menu  */

     if (menuid != 0)
     {
          _settextcolor(11) ;
          _settextposition(15,30) ;
          _outtext(menuchoice) ;

/*       Blank menu area */

          _settextposition(16,25) ;
          _outtext(blnkbuf) ;
          _settextposition(17,25) ;
          _outtext(blnkbuf) ;
          _settextposition(18,25) ;
          _outtext(blnkbuf) ;
          _settextposition(19,25) ;
          _outtext(blnkbuf) ;
          _settextposition(20,25) ;
          _outtext(blnkbuf) ;
          _settextposition(21,25) ;
          _outtext(blnkbuf) ;

          if (menuid == 1)
          {
```

```
/*          Menu 1 text          */

                _settextposition(16,25) ;
                _outtext(menu1a) ;
                _settextposition(17,25) ;
                _outtext(menu1b) ;
                _settextposition(18,25) ;
                _outtext(menu1c) ;
                _settextposition(19,25) ;
                _outtext(menu1d) ;
                _settextposition(20,25) ;
                _outtext(menu1e) ;
                _settextposition(21,25) ;
                _outtext(menu1f) ;

           }
           else
           {

/*          Menu 2 choice */

                _settextposition(16,25) ;
                _outtext(menu2a) ;
                _settextposition(17,25) ;
                _outtext(menu2b) ;
                _settextposition(18,25) ;
                _outtext(menu2c) ;
                _settextposition(19,25) ;
                _outtext(menu2d) ;
                _settextposition(20,25) ;
                _outtext(menu2e) ;
           }
      }

/*  If value display is requested */
/*  Then    */
/*      Display values if requested */

      if (valflg != 0)
      {
           _settextcolor(12) ;
           _settextposition(8,50) ;
           specchar = 0xb0 ;
           sprintf(buffer,"                          ") ;
           strset(buffer,specchar) ;
           _outtext(buffer) ;
           _settextposition(10,50) ;
           _outtext(buffer) ;
           _settextposition(13,50) ;
           _outtext(buffer) ;
```

```
            sprintf(buffer,"  ") ;
            strset(buffer,specchar) ;
            _settextposition(11,50) ;
            _outtext(buffer) ;

            if (trans_tbl[disprow].x25addr[0] != '\0')
            {
                 sprintf(buffer,"%s",trans_tbl[disprow].x25addr) ;
                 _settextposition(8,50) ;
                 _outtext(buffer) ;
            }

            if (trans_tbl[disprow].tcpaddr[0] != '\0')
            {
                 sprintf(buffer,"%s",trans_tbl[disprow].tcpaddr) ;
                 _settextposition(10,50) ;
                 _outtext(buffer) ;
                 sprintf(buffer,"%d",trans_tbl[disprow].tcpport) ;
                 _settextposition(11,50) ;
                 _outtext(buffer) ;

            }

            if (trans_tbl[disprow].isdnaddr[0] != '\0')
            {
                 sprintf(buffer,"%s",trans_tbl[disprow].isdnaddr) ;
                 _settextposition(13,50) ;
                 _outtext(buffer) ;
            }
       }

}
```

```
char    errmsg[] = "Improper Key Enterred, Valid keys A thru F" ;
char    err2msg[] = "Address not found in table, Please re-enter" ;
char    err3msg[] = "No empty rows are currently available" ;

ushort DET_ED_ACT(char edkey, ushort edrow)

{

/* This routine determines the required edit action from the enterred
key */

   char    search_strng[22] ;
   ushort  ii ;


/*  Select Search Criteria Based on enterred key */
/*      CASE create new entry  */

   edkey = toupper(edkey) ;
   switch(edkey)
   {
      case 'A' :

/*          Determine next blank row in table */
/*          Edit the new row                    */

        for(ii = 0; ii < MAX_TBL &&
            trans_tbl[ii].isdnaddr[0] != '\0' ; ii++ ) ;
        if (ii < MAX_TBL)
        {
          ED_SEL_ROW(ii) ;
          edrow = ii ;
        }
        else
        {
          output_message(err3msg) ;
        }
        break ;

/*      CASE edit current entry */
/*          Edit current row */

     case 'B' :

        ED_SEL_ROW(edrow) ;
        break ;

/*      CASE edit specified X.25 address */
/*          Request X.25 address to find */
/*          Find address in the table */
```

```
/*          Edit specified row */

     case 'C' :

        _settextposition(24,10) ;
        printf("Enter X.25 Address Desired (followed by <CR>) ") ;

        search_strng[0] = '\0' ;
        GET_INPUT_ADDR(X25TYP,24,57,search_strng,X25_ADDR_LEN) ;

        _settextposition(24,10) ;
        printf("
            ");

        for(ii = 0; ii < MAX_TBL &&
               strcmp(search_strng,trans_tbl[ii].x25addr) != 0;ii++);

        _settextposition(24,10) ;
        printf("search: %s , %s , %d",search_strng,
                     trans_tbl[ii].x25addr,ii) ;

        if (ii < MAX_TBL)
        {
           ED_SEL_ROW(ii) ;
           edrow = ii ;
        }
        else
        {
           output_message(err2msg) ;
        }
        break ;

/*      CASE edit specified TCP address */
/*          Request TCP address to find */
/*          Find address in the table */
/*          Edit specified row */

     case 'D' :

        _settextposition(24,10) ;
        printf("Enter TCP Address Desired (followed by <CR>) ") ;

        search_strng[0] = '\0' ;
        GET_INPUT_ADDR(TCPTYP,24,57,search_strng,TCP_ADDR_LEN) ;

        _settextposition(24,10) ;
        printf("
            ");

        for(ii = 0; ii < MAX_TBL &&
```

```
                strcmp(search_strng,
                 trans_tbl[ii].tcpaddr) != 0;ii++);

          if (ii < MAX_TBL)
          {
             ED_SEL_ROW(ii) ;
             edrow = ii ;
          }
          else
          {
             output_message(err2msg) ;
          }
          break ;

/*        CASE scroll to next entry */
/*            Increment current row number */
/*            Display next row */

        case 'E' :
          edrow++ ;
          if (edrow == MAX_TBL)
          {
             edrow = 0 ;
          }
          DISP_SCRN(edrow,0,0,1) ;
          break ;

/*        CASE Exit */
/*            no-op */

        case 'F' :
          break ;

/*        DEFAULT */
/*            Indicate improper character enterred */

        default :
          output_message(errmsg) ;
          break ;

       }
     return(edrow) ;
  }
```

```
char    illchar[] = "Illegal character enterred for this address
position" ;

char    GET_INPUT_ADDR(ushort addrtyp,ushort inprow,ushort inpcol,
              char *inp_buffer, ushort num_inp)

{

/*  This routine gets an input address from the specified location on
*/
/*      screen and stores in the given buffer */

   char    inpkey,specchar ;
   ushort  numchar ;


/*  Loop until a <CR> or <ESC> has been received */

   inpkey = 0 ;

   numchar = 0 ;

   while (inpkey != 0x0d && inpkey != ESCAPE && inpkey != DELKEY &&
          numchar < num_inp )
   {

/*      Wait for user input */

      inpkey = WAIT_USER_INPUT() ;

/*      If the character is valid for this type of address in this
position */
/*      Then */
/*          Add current character to buffer */
/*          Display current buffer on the screen */

      if (strchr(numset,inpkey) != NULL ||
          (addrtyp == TCPTYP && inpkey == '.'))
      {

         if (numchar == 0)
         {
            _settextposition(inprow,inpcol) ;
            specchar = 0xb0 ;
            sprintf(buffer,"                    ") ;
            strnset(buffer,specchar,num_inp) ;
            _outtext(buffer) ;
         }
         _settextposition(inprow,inpcol) ;
         sprintf(inp_buffer,"%s%c",inp_buffer,inpkey) ;
```

```
            putc(inpkey,stdout) ;
            inpcol++ ;
            numchar++ ;
         }
/*       else */
/*            If character was not a <CR> or <ESC> or <DEL> */
/*            Then */
/*                 Indicate illegally enterred character */

         else
         {
            if (inpkey != 0x0d && inpkey != ESCAPE && inpkey != DELKEY )
            {
               output_message(illchar) ;
            }
         }
      }

      if (inpkey == 0x0d)
      {
         inpkey = '\0' ;
      }

      return (inpkey) ;
}
```

```
void    ED_SEL_ROW(ushort selrow)
{

/* This routine edits a selected row in the translation table */

   char    exitkey ;


/*  Re-display screen values and menu for edit session */

   DISP_SCRN(selrow,2,0,1) ;

/*  Loop until <ESC> character encountered */

   exitkey = '\0' ;
   while (exitkey != ESCAPE )
   {

/*      Accept updates to the X.25 Address  */

      exitkey = UP_X25_ADDR(selrow) ;

/*      IF the user has not indicated he has completed */
/*      Then */
/*          Accept updates to the TCP address */

      if (exitkey != ESCAPE)
      {
         exitkey = UP_TCP_ADDR(selrow) ;
      }

/*      IF the user has not indicated he has completed */
/*      Then */
/*          Accept updates to the ISDN address */

      if (exitkey != ESCAPE)
      {
         exitkey = UP_ISDN_ADDR(selrow) ;
         if (trans_tbl[selrow].isdnaddr[0] != '\0')
         {
            entry_exists = 1 ;
         }
      }
   }
/*  Re-display screen values & enclosing menu for next session */

   DISP_SCRN(selrow,1,0,1) ;
}
```

```
char    x25error[] = "Address Already in table!!!" ;

char    UP_X25_ADDR(ushort uprow)
{
/*  This routine accepts updates to the X25 address field of the */
/*      Address translation table   */

   ushort  legaddr, ii ;
   char    inpkey ;
   char    addr_buffer[21] ;

/*  Loop until legal address received */

   legaddr = 0 ;
   while (legaddr == 0)
   {

/*      Get input address from its field on the screen */
/*      If an address has been enterred */
/*      Then */

      addr_buffer[0] = '\0' ;

      inpkey = GET_INPUT_ADDR(X25TYP,8,50,addr_buffer,X25_ADDR_LEN) ;

      if (addr_buffer[0] != '\0' && inpkey != DELKEY)
      {

/*          Search table for matching address not in this row */

         for (ii=0 ; (strcmp(addr_buffer,trans_tbl[ii].x25addr) != 0 ||
                ii == uprow) && ii < MAX_TBL ; ii++ ) ;

/*          If entry was found */
/*          Then */
/*              Indicate entry is already present */

         if ( ii < MAX_TBL )
         {
            output_message(x25error) ;
         }
         else
         {
            legaddr = 1 ;
            strcpy(trans_tbl[uprow].x25addr,addr_buffer) ;
         }
      }

/*      else */
/*          if the enterred key was a delete key */
```

```
/*          then */
/*              delete the current field entry */

      else
      {
         if (inpkey == DELKEY)
         {
            trans_tbl[uprow].x25addr[0] = '\0' ;
            inpkey = '\0' ;
         }
         legaddr = 1 ;
      }
   }
   return(inpkey) ;
}
```

```
char    errval[] = "Address subsection value must be less than 255" ;
char    errport[] = "Port id must be less than 99" ;

char    UP_TCP_ADDR(ushort up2row)
{
/*  This routine accepts updates to the X25 address field of the */
/*      Address translation table   */

   ushort  legaddr, ii, errfnd, subsect,portval ;
   char    inpkey ;
   char    addr_buffer[21],addr_subs[4] ;

/*  Loop until legal address received */

   legaddr = 0 ;
   while (legaddr == 0)
   {

/*      Get input address from its field on the screen */
/*      If an address has been enterred */
/*      Then */

     addr_buffer[0] = '\0' ;

     inpkey = GET_INPUT_ADDR(TCPTYP,10,50,addr_buffer,TCP_ADDR_LEN) ;

     if (addr_buffer[0] != '\0' && inpkey != DELKEY)
     {

/*          Convert the address to displayable format */

       CONV_TCP_RX(CONV_TCP_TX(addr_buffer),addr_buffer) ;

/*          Copy the address to the translation table */

       legaddr = 1 ;
       strcpy(trans_tbl[up2row].tcpaddr,addr_buffer) ;

     }

/*      else */
/*          if the enterred key was a delete key */
/*          then */
/*              delete the current field entry */

     else
     {
       if (inpkey == DELKEY)
       {
         trans_tbl[up2row].tcpaddr[0] = '\0' ;
```

```
            inpkey = '\0' ;
          }
          legaddr = 1 ;
        }
      }

/*  If legal address entered and key was not <DEL> */
/*  Then */
/*       Loop until legal row number received */

      if (legaddr == 1 && inpkey != DELKEY)
      {
        legaddr = 0 ;
        while (legaddr == 0)
        {

/*            Input the port number */
/*            Convert the input string to integer */

          addr_buffer[0] = '\0' ;
          GET_INPUT_ADDR(X25TYP,11,50,addr_buffer,2) ;
          portval = atoi(addr_buffer) ;

/*            If the input port is in the proper range */
/*            Then */
/*                Store the input port in the translation table */

          if (portval >= 0 && portval <= 99)
          {
            legaddr = 1 ;
            trans_tbl[up2row].tcpport = portval ;
          }
          else
          {
            output_message(errport) ;
          }
        }
      }

      return(inpkey) ;
    }
```

```
char    UP_ISDN_ADDR(ushort up3row)
{
/*  This routine accepts updates to the ISDN address field of the */
/*      Address translation table   */

   ushort  legaddr, ii ;
   char    inpkey ;
   char    addr_buffer[21] ;

/*  Loop until legal address received */

   legaddr = 0 ;
   while (legaddr == 0)
   {

/*      Get input address from its field on the screen */
/*      If an address has been enterred */
/*      Then */

      addr_buffer[0] = '\0' ;

      inpkey = GET_INPUT_ADDR(ISDNTYP,13,50,addr_buffer,ISDN_ADDR_LEN)
;

      if (addr_buffer[0] != '\0' && inpkey != DELKEY)
      {

         legaddr = 1 ;
         strcpy(trans_tbl[up3row].isdnaddr,addr_buffer) ;
      }

/*       else */
/*           if the enterred key was a delete key */
/*           then */
/*               delete the current field entry */
/*           else */
/*               if the address field is currently blank */
/*               then */
/*                   set the address to the local address */

      else
      {
         if (inpkey == DELKEY)
         {
            trans_tbl[up3row].isdnaddr[0] = '\0' ;
            inpkey = '\0' ;
         }
         else
         {
            if (trans_tbl[up3row].isdnaddr[0] == '\0')
```

```
                {
                   strcpy(trans_tbl[up3row].isdnaddr,loc_isdn_addr) ;
                }
             }
             legaddr = 1 ;
          }
       }
       return(inpkey) ;
    }
```

```
void    STORE_TRANS_TBL(ushort exist_flag)
{

/*  This routine stores the current version of the translation table
*/
/*      into the special gateway file GATTRANS.TBL  */

   FILE    *curfil ;
   int     ii ;


/*  If an entry exists in the table */
/*  Then */
/*      Open the output file for writing */

   if (exist_flag != 0)
   {
      curfil = fopen("GATTRANS.TBL","w") ;

/*      If the file was opened successfully */
/*      Then */
/*          Loop for all entries in the table */

      if (curfil != NULL)
      {
         for(ii=0; ii < MAX_TBL ; ii++)
         {

/*              if the current entry is not blank */
/*              then */
/*                  store the entry to the table */

            if (trans_tbl[ii].isdnaddr[0] != '\0')
            {
               fprintf(curfil,"%s,%s,%d,%s\n",
                  trans_tbl[ii].x25addr,
                  trans_tbl[ii].tcpaddr,
                  trans_tbl[ii].tcpport,
                  trans_tbl[ii].isdnaddr) ;
            }
         }
         fclose(curfil) ;
      }
   }
}
```

```
ushort  READ_TRANS_TBL()
{

/*  This routine reads the stored values of the translation table from
*/
/*      the hard disk and initializes the table */

   int      ii, curress ;
   FILE     *rdptr ;
   char     tmp_strng[10] ;

/*  Initialize the table ot all nulls */

   for (ii = 0 ; ii < MAX_TBL ; ii++)
   {
      trans_tbl[ii].x25addr[0] = '\0' ;
      trans_tbl[ii].x25ptr = 0 ;
      trans_tbl[ii].tcpaddr[0] = '\0' ;
      trans_tbl[ii].tcpport = 0 ;
      trans_tbl[ii].tcpptr = 0 ;
      trans_tbl[ii].isdnaddr[0] = '\0' ;
   }

/*  Open the disk file for read access */
/*  If the file opens successfully */
/*  Then */
/*      Loop for all entries in the file */

   if ((rdptr = fopen("GATTRANS.TBL","r")) != NULL)
   {
      entry_exists = 1 ;
      for (ii = 0; ii < MAX_TBL && curress != EOF ; ii++)
      {

/*          Read the current entry */
/*          If the entry is not blank */
/*          Then */
/*              Indicate that at least 1 entry exists in the table */

         curress = fscanf(rdptr,"%[^, ],%[^, ],%[^, ],%s \n",
                     trans_tbl[ii].x25addr,trans_tbl[ii].tcpaddr,
                     tmp_strng,trans_tbl[ii].isdnaddr) ;
         trans_tbl[ii].tcpport = atoi(tmp_strng) ;
      }
      fclose(rdptr) ;
   }

   return(entry_exists) ;
}
```

```
void    DISP_BRK_STATS(ushort linktyp,STAT_ENTRY *cur_entry)
{

/*  This routine displays accumulated statistics for a particular link
type */

   long curtime;
   char dest1[8],dest2[8] ;


/*  Get the current time */
/*  Determine destination and print destination types */
/*  Display the formatted header line */

   time(&curtime);
   _settextcolor (15);

   if(starttime>0) {_settextposition(5,60);
         sprintf(obuf,"Up Time: %d Mins",(curtime-starttime)/60) ;
         _outtext(obuf);
        }

   _settextposition (7,14);
   sprintf(dest1,"[X.25]") ;
   sprintf(dest2,"[ISDN]") ;
   if (linktyp == X25TYP)
   {
      sprintf(dest1,"[TCP]") ;
   }
   else
   {
      if (linktyp == ISDNTYP)
      {
         sprintf(dest2,"[TCP]") ;
      }
   }
   printf("Remote 1 %s                        Remote 2 %s",dest1,dest2);

/*  Print statistics background */

   _settextposition(9,14) ;
   printf("Calls Proc'd :                       Calls Proc'd :\n");
   _settextposition(10,14) ;
   printf("Packets Recv :                       Packets Recv : \n");
   _settextposition(11,14) ;
   printf("Bytes Recv   :                       Bytes Recv   : \n");
   _settextposition(12,14) ;
   printf("Packets Sent :                       Packets Sent : \n");
   _settextposition(13,14) ;
   printf("Bytes Sent   :                       Bytes Sent   : \n");
```

```
_settextposition(9,29);sprintf(obuf,"%d",cur_entry->rem1_call_count);
   _outtext(obuf);

_settextposition(9,63);sprintf(obuf,"%d",cur_entry->rem2_call_count) ;
   _outtext(obuf);

_settextposition(10,29);sprintf(obuf,"%d",cur_entry->rem1_packets_recv
);
   _outtext(obuf);

_settextposition(10,63);sprintf(obuf,"%d",cur_entry->rem2_packets_recv
) ;
   _outtext(obuf);

_settextposition(11,29);sprintf(obuf,"%ld",cur_entry->rem1_bytes_recv)
;
   _outtext(obuf);

_settextposition(11,63);sprintf(obuf,"%ld",cur_entry->rem2_bytes_recv)
;
   _outtext(obuf);

_settextposition(12,29);sprintf(obuf,"%d",cur_entry->rem1_packets_sent
);
   _outtext(obuf);

_settextposition(12,63);sprintf(obuf,"%d",cur_entry->rem2_packets_sent
) ;
   _outtext(obuf);

_settextposition(13,29);sprintf(obuf,"%ld",cur_entry->rem1_bytes_sent)
;
   _outtext(obuf);

_settextposition(13,63);sprintf(obuf,"%ld",cur_entry->rem2_bytes_sent)
;
   _outtext(obuf);
}
```

```
char    tcp_title[] = "TCP STATISTICS SCREEN\n" ;

void TCP_STATS()
{

/*  This routine displays the current set of statistics for the TCP
link */
   UCMD    *tcp_stat_ptr,tcp_stat ;
   int ii,stat_stat ;
   char * cptr ;

/*  Display the number of calls processed */

   _clearscreen(_GCLEARSCREEN);
   _settextposition(15,15) ;
   printf("Total # of calls processed : %d \n",tcp_call_count) ;

/*  Display the rest of the statistics by destination */

   DISP_BRK_STATS(TCPTYP,&brk_stats[TCPTYP]) ;

/*  Fill out the constant portion of the TCB */

   tcp_stat_ptr = &tcp_stat ;
   cptr = (char *) &tcp_stat.tcp_pcb ;

   for(ii=0; ii < sizeof(tcp_stat.tcp_pcb) ; ii++,cptr++ )
   {
      *cptr = '\x00' ;
   }

   tcp_stat.tcp_pcb.p_op = OP_STATUS ;
   tcp_stat.tcp_pcb.p_protocol = PROTO_TCP ;

/*  Loop for all of the allowable outstanding messages */

   /* Debug */


   for(ii=0 ; ii < MAX_TCP_OUT ; ii++)
   {

/*       If the current connection is outstanding */
/*       Then */
/*           Display the screen background and wait for input */

      if (tcp_out[ii].int_call_typ != NOT_CONNECTED)
      {
         stat_scr(tcp_title) ;
         _clearscreen(_GCLEARSCREEN) ;
```

```
/*          Fill out the connection part of the TCP PCB */
/*          Issue the status command */

        tcp_stat.tcp_pcb.p_id = tcp_out[ii].cur_typ_id ;
        stat_stat = TCPCTL(&tcp_stat.tcp_pcb) ;

/*          If error not encountered */
/*          Then */
/*              Format and display the protocol statistics */

        if (stat_stat != -1)
        {

           tcp_stats_rx = 0 ;
           while (tcp_stats_rx == 0)
           {
              QUE_TCP_ACT(&tcp_stat) ;
           }

           _settextposition(7,28) ;
           sprintf(obuf,"Active Connection: Id # %d",
                              tcp_out[ii].cur_typ_id) ;
           _outtext(obuf) ;
           _settextcolor(15) ;
           _settextposition(8,27) ;
           sprintf(obuf,"Connection Status: %s",
                     tcp_stat_typ[tcp_stat.tcp_pcb.p_id]) ;
           _outtext(obuf) ;
           _settextposition(10,10) ;
           sprintf(obuf,"Local Node Address   :  %20s          Port ID:
%d",
                     trans_tbl[tcp_out[ii].cur_typ_entry].tcpaddr,
                     tcp_stat.tcp_pcb.p_fport) ;
           _outtext(obuf) ;
           _settextposition(11,10);
           sprintf(obuf,"Foreign Node Address :  %20s          Port ID:
%d",
                     trans_tbl[tcp_out[ii].tar_typ_entry].tcpaddr,
                     tcp_stat.tcp_pcb.p_lport) ;
           _outtext(obuf) ;
           _settextposition(13,25) ;
           sprintf(obuf,"Transmit Control Word : %ld",
                              tcp_stat.tcp_pcb.p_tcc) ;
           _outtext(obuf) ;
           _settextposition(14,25);
           sprintf(obuf,"Transmit Timeout Value: %d",
                              tcp_stat.tcp_pcb.p_retrantimeout) ;
           _outtext(obuf) ;
           _settextposition(16,28);
```

```
            sprintf(obuf,"Bytes to Receive : %d",
                                tcp_stat.tcp_pcb.p_rcv_cnt) ;
            _outtext(obuf) ;
            _settextposition(17,28);
            sprintf(obuf,"Bytes to Transmit : %d",
                                tcp_stat.tcp_pcb.p_snd_cnt) ;
            _outtext(obuf) ;
         }
      }
   }

/*  Display the screen background and wait for input */

   stat_scr(tcp_title) ;


}
```

```
char x25notup[] =   "X25 Link not initialized, Enter option A" ;
char isdnnotup[] =  "ISDN Link not initialized, Enter option B" ;
char tcpnotup[] = "TCP link not initialized, Enter option C" ;

void SETUP_GATE(ncb_t far *sxncb,ushort x25flg,ushort isdnflg,ushort
tcpflg)
{

/*  This routine sets up all 3 links to listen for incoming calls.
Setup */
/*  is conditional on previous entry of the entry screen
initialization */
/*  commands */

   ushort  errflg,i ;

/*  Initialize the outstanding calls tables */

   for(i = 0; i < MAX_X25_OUT ; i++)
   {
      x25_out[i].int_call_typ = NOT_CONNECTED ;
   }
   for(i = 0; i < MAX_ISDN_OUT ; i++)
   {
      isdn_out[i].int_call_typ = NOT_CONNECTED ;
   }
   for(i = 0; i < MAX_TCP_OUT ; i++)
   {
      tcp_out[i].int_call_typ = NOT_CONNECTED ;
   }


/*  If the X.25 link has been initialized */
/*  Then */
/*      Setup the X.25 link to listen */

   if (x25flg == 1)
   {
      INIT_X25_LISTEN() ;
   }
   else
   {
      output_message(x25notup) ;
   }

/*  If the isdn link has been initialized */
/*  Then */
/*      Setup the ISDN link to get the DPP LSN */

   if (isdnflg == 1)
```

```
   {
      isdn_x25d() ;

/*       Initialize the SIP to get its LSN */

      isdn_sip() ;

/*       Start the X.25 DPP to listen */

      isdn_start_dpp() ;

/*       Initiate the first wait for DPP data */

      if (isdn_call_wait() == 0)
      {
         sprintf(obuf,"isdn wait error # %hx",ircvncb.ncb_retcode) ;
         output_message(obuf) ;
      }
   }
   else
   {
      output_message(isdnnotup) ;
   }

/*  If the TCP link has been initialized */
/*  Then */
/*       Initialize the user settable parameters (Hook only) */

   if (tcpflg == 1)
   {
      errflg = SETUP_TCP_GATEWAY() ;

/*       If no error encountered */
/*       Then */
/*           Start listening on the link */

      if (errflg == 0)
      {
         INIT_TCP_LISTEN() ;
      }
   }
   else
   {
      output_message(tcpnotup) ;
   }

/*  Setup Global gateway variables */

   gateway_up = 1 ;
   time(&starttime) ;       }
```

```
/*********************************************************/
/*** The following code and include files make up the***/
/*** Edit and Statistics Handling tasks of the MMI   ***/
/*** module                                          ***/
/*** This file is ED&STATS.C                         ***/
/*********************************************************/



#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <graph.h>
#include <math.h>

/* FRONTIER/TCP CONSTANT DECLARATIONS */

#include "c:\tcp\include\ftctypes.h"
#include "c:\tcp\include\ulpuser.h"
#include "c:\tcp\include\ulpshare.h"

/*  GATEWAY CONSTANT DECLARATIONS */

#include "c:\devel\gatedef.h"

/* GATEWAY TYPE DEFINITIONS */

#include "c:\devel\tipedef.h"

/* EXTERNAL VARIABLE DECLARATIONS */

#include "c:\devel\globaldef.h"

/* SUBROUTINE CALL LIST DECLARATIONS */

#include "c:\devel\subdecl.h"


/*********************************************************/
/***              Statistics Handling Task             ***/
/*********************************************************/

char    gate_st_title[] = "GATE STATISTICS SCREEN\n" ;
/*
This routine displays the overall gateway transmission statistics
*/
void trans_stats()
{
   int i,rown,coln;
```

```
/*  Display the ISDN and X25 stats using the common format */
   _clearscreen(_GCLEARSCREEN);

   DISP_BRK_STATS(TCPTYP,&ovr_stats) ;

   _settextposition (14,30) ;
   printf("Remote 3 [TCP]");
   _settextposition(15,30) ;
   printf("Calls Proc'd :");
   _settextposition(16,30) ;
   printf("Packets Recv : ");
   _settextposition(17,30) ;
   printf("Bytes Recv   : ");
   _settextposition(18,30) ;
   printf("Packets Sent : ");
   _settextposition(19,30) ;
   printf("Bytes Sent   : ");

   _settextposition(15,46);
   sprintf(obuf,"%d",tcp_call_count);
   _outtext(obuf);
   _settextposition(16,46);
   sprintf(obuf,"%d",tcp_packets_recv);
   _outtext(obuf);
   _settextposition(17,46);
   sprintf(obuf,"%ld",tcp_bytes_recv);
   _outtext(obuf);
   _settextposition(18,46);
   sprintf(obuf,"%d",tcp_packets_sent);
   _outtext(obuf);
   _settextposition(19,46);
   sprintf(obuf,"%ld",tcp_bytes_sent);
   _outtext(obuf);
   if (tot_calls != 0 && tot_packs != 0)
   {
     sprintf(obuf,"Average Queue delay: Calls - %ld msec , Data - %ld
msec",
                  call_queued_time/tot_calls,
                  pack_queued_time/tot_packs) ;
     _settextposition(20,10);
     _outtext(obuf);
     sprintf(obuf,"Average Processing time: Calls - %ld msec , Data -
%ld msec",
                  call_process_time/tot_calls,
                  pack_process_time/tot_packs) ;
     _settextposition(21,10);
     _outtext(obuf);
   }
```

```
   stat_scr(gate_st_title) ;

   _clearscreen(_GCLEARSCREEN);
   _settextposition(7,9);
   printf("Remote 1 [x.25] Last Packet Received :\n");

   for(i=0;i<lpx.dm.data_Size && i < 88;i++)
   {
      rown = i/22 ;
      coln = 3*(i - (rown * 22)) ;
      _settextposition(8+rown,9+coln) ;
      printf("%hx ",lpx.dm.data_Data[i]);
   }

   _settextposition(12,9);
   printf("Remote 2 [ISDN] Last Packet Received :\n");
   for(i=0;i<lpi.dm.data_Size && i < 88;i++)
   {
      rown = i/22 ;
      coln = 3*(i - (rown * 22)) ;
      _settextposition(13+rown,9+coln) ;
      printf("%hx ",lpi.dm.data_Data[i]);
   }

   _settextposition(17,9);
   printf("Remote 3 [TCP] Last Packet Received :\n");
   for(i=0;i<tcp_lpr_size && i < 88;i++)
   {
      rown = i/22 ;
      coln = 3*(i - (rown * 22)) ;
      _settextposition(18+rown,9+coln) ;
      printf("%hx ",tcplpr[i]);
   }

   stat_scr(gate_st_title);
}

char    x25_title[] = "X.25 STATISTICS SCREEN\n" ;

/*
This routine displays the X.25 specific statistics requests
*/
void x25_stats()
{
   int i,ii;
   ncb_t ncb;
   ncb_t far *sncb;
   struct xstat_record
   {
```

```
    ushort  reserve;
    ushort  lcn;
    uchar   session;
    uchar   status;
    char    aname[16];
    char    pname[16];
    uchar   rncbp;
    uchar   sncbp;
    char    DTE[16];
    ulong   npos;
    ulong   npya;
    ulong   ncos;
    ulong   ncya;
    ulong   nprn;
    ulong   npyr;
    ulong   ncrn;
    ulong   ncyr;
    ulong   nrDTE;
    ulong   nrnet;
  }xstat;

  _clearscreen(_GCLEARSCREEN) ;
  DISP_BRK_STATS(X25TYP,&brk_stats[X25TYP]) ;
  stat_scr(x25_title) ;

  for (ii=0 ; ii < MAX_X25_OUT ; ii ++)
  {

    if (x25_out[ii].int_call_typ != NOT_CONNECTED)
    {
      clear_ncb(sncb);
      sncb->ncb_command = 0x34;
      sncb->ncb_lsn = x25_out[ii].cur_typ_id ;
      sncb->ncb_buffer = (char *) &xstat;
      sncb->ncb_length = sizeof(xstat);
      sncb->ncb_lana_num = 0xff;
      srcp = X25_NAME;
      destp = sncb->ncb_callname;
      for(i=0;i<NB_NAME_LEN;i++,srcp++,destp++)
        *destp = *srcp;
      send_cmd_5c(sncb);

      _clearscreen(_GCLEARSCREEN);
      _settextcolor (15);
      _settextposition(8,17);
      printf("Session number                    :
%d\n",xstat.session);
      printf("\t\tSession status                  : ");
      switch (xstat.status)
      {
```

```
            case 1 :
               printf("LISTEN pending\n");
               break;
            case 2 :
               printf("CALL pending\n");
               break;
            case 3 :
               printf("Session connected\n");
               break;
            case 4 :
               printf("HANGUP pending\n");
               break;
            case 5 :
               printf("HANGUP complete\n");
               break;
            case 6 :
               printf("Circuit Cleared\n");
               break;
            default :
               printf("Circuit not Established\n") ;
               break ;
         }
         printf("\t\tNumber of Receive NCBs pending    :
%d\n",xstat.rncbp);
         printf("\t\tNumber of Send    NCBs pending    :
%d\n",xstat.sncbp);
         printf("\t\tUser packets offered/not acked    : %d/%d\n",
            xstat.npos,xstat.npya);
         printf("\t\tUser characters offered/not acked : %d/%d\n",
            xstat.ncos,xstat.ncya);
         printf("\t\tNetwork packets offered/not read  : %d/%d\n",
            xstat.nprn,xstat.npyr);
         printf("\t\tNetwork character offered/not read: %d/%d\n",
            xstat.ncrn,xstat.ncyr);
         printf("\t\tNumber of resets by local DTE     : %d\n",
            xstat.nrDTE);
         printf("\t\tNumber of resets by the Network   : %d\n",
            xstat.nrnet);
         stat_scr(x25_title);
      }
   }
}

/*
This routine displays the status screen template which is
common to all screens and then waits for user input to continue
*/
void stat_scr(char *givn_titl)
{
   struct videoconfig vc;
```

```
   long int j = 7;

/*  ADD TOP RECTANGLE */

   _setcolor(15);
   _settextcolor (10);
   _settextposition (5,30);
   sprintf(obuf,"%s",givn_titl);
   _outtext(obuf);

   _setlinestyle(0xFFFF);
   _rectangle(_GBORDER, -300,0,300,40);
   _setcolor(9);
   _rectangle(_GBORDER, -299,1,299,39);

/*  ADD TEXT */

   _settextcolor (10);
   _settextposition (2,27) ;
   _outtext (gatetitle) ;

   _rectangle(_GBORDER, -300,44,300,296);
   _rectangle(_GBORDER, -299,45,299,295);

/* ADD LOWER RECTANGLE */

   _setcolor(7);
   _setlinestyle(0xFFFF);
   _rectangle(_GBORDER, -300,300,300,340);
   _rectangle(_GBORDER, -299,301,299,339);

   sprintf(obuf,"Hit any key to proceed.\n");
   output_message(obuf);
   WAIT_USER_INPUT();


}

/*
This routine displays the TCP specific statisitcs
*/
#include <c:\devel\tcp_stat.c>

/*
This routine displays the ISDN specific statisitcs
*/
#include <c:\devel\isdn_stat.c>

/*
This routine displays common portions of the statistics information
```

```
*/
#include <c:\devel\di_br_st.c>


/*************************************************************/
/***                  Edit Handling Task                   ***/
/*************************************************************/

/*
This routine oversees the translation table edit session
*/
#include <c:\devel\ed_tr_tb.c>

/*
This routine displays the current edit screen, menu and values
*/
#include <c:\devel\dispscrn.c>

/*
This routines determines and initiates the operator requested action
*/
#include <c:\devel\de_ed_ac.c>



/*
This routine performs the general operations required to
input an address from the operator
*/
#include <c:\devel\ge_in_ad.c>

/*
This routine oversees the edit of a selected row
*/
#include <c:\devel\ed_se_rw.c>

/*
This routine performs the X.25 specific actions required to
input an X.25 address
*/
#include <c:\devel\up_x2_ad.c>

/*
This routine performs the TCP specific actions required to
input a TCP address
*/
#include <c:\devel\up_tc_ad.c>

/*
This routine performs the ISDN specific actions required to
```

```
input an ISDN address
*/
#include <c:\devel\up_is_ad.c>

/*
This routine stores the current translation table at system shutdown
*/
#include <c:\devel\st_tr_tb.c>

/*
This routine reads the previous contents of the translation
table at system startup
*/
#include <c:\devel\rd_tr_tb.c>
```

```
void    INIT_TCP_LISTEN()
{
/* This routine sets up the TCP links for listening */

int ii ;

/*  Loop for all translation table entries */

   for(ii=0; ii<= MAX_TBL ; ii++)
   {

/*       If the current address set is not on the local end of the ISDN
or */
/*           there is a X.25 address assigned to this TCP address */
/*       Then */

     if (trans_tbl[ii].tcpaddr[0] != '\0' &&
          (strcmp(trans_tbl[ii].isdnaddr,loc_isdn_addr) != 0 ||
          (trans_tbl[ii].x25addr[0] != '\0')))
     {

/*           allocate a TCP listen buffer for this link */

        trans_tbl[ii].tcpptr = (PCB *) malloc(sizeof(PCB)) ;

/*           Initiate the listen request for this link   */

        TCP_LISTEN(ii) ;

/*           display action */

        if (gattrace == 1 || gattrace == 3)
        {
          sprintf(obuf,"Listen initiated for TCP link: %s",
                  trans_tbl[ii].tcpaddr) ;
          output_message(obuf) ;
        }
     }
   }
   sprintf(obuf,"Gate Setup Complete") ;
   output_message(obuf) ;
}
```

```
char tcp_listen_error[] = "Error initiating TCP listen" ;

void TCP_LISTEN(int rownum)
{

/* This routine initiates a listen on a specified TCP link */

   PCB locpcb ;
   PCB far* stnp = &locpcb ;
   int ii,stat_list ;
   char far * cptr ;

/*  Clear the listen pcb */

   cptr = (char far *) trans_tbl[rownum].tcpptr ;
   for(ii=0; ii < sizeof(*trans_tbl[rownum].tcpptr); ii++,cptr++)
   {
      *cptr = '\x00' ;
   }

/*  Fill out the required fields of the pcb */

   trans_tbl[rownum].tcpptr->p_op = OP_LISTEN ;
   trans_tbl[rownum].tcpptr->p_protocol = PROTO_TCP ;
   trans_tbl[rownum].tcpptr->p_lport = trans_tbl[rownum].tcpport ;

/*  Convert the TCP address to the transmittable format */

   trans_tbl[rownum].tcpptr->p_lhost =
         CONV_TCP_TX(trans_tbl[rownum].tcpaddr) ;
   trans_tbl[rownum].tcpptr->p_pid = rownum ;

/*  Initiate the listen */

   stat_list = TCPCTL(trans_tbl[rownum].tcpptr) ;

/*  If an error was encountered */
/*  Then */
/*       Indicate error message */

   if (stat_list != 0)
   {
      output_message(tcp_listen_error) ;
   }
}
```

```
U32BIT  CONV_TCP_TX(char *strngptr)
{
/* This routine converts the string version of a TCP address to its */
/*      Network transmittable format */

   U32BIT  fmtbuffer = 0 ;
   int     ii,posptr = 0,curpos = 0, intval ;
   ushort  tmpnum ;
   char    tmpstrng[5] ;


/*  Loop for all 4 subsections of the address */
/*      Find the end of the current subsection */

   posptr = 0 ;

   for(ii=0 ; ii<4 ; ii++)
   {
      fmtbuffer = fmtbuffer << 8 ;
      strngptr += posptr ;
      posptr = strcspn(strngptr,".") ;

      if (posptr == NULL)
      {
         posptr = 3 ;
      }

      strncpy(tmpstrng,strngptr,posptr) ;
      tmpstrng[posptr] = '\0' ;
      posptr++ ;

/*       Convert the subsection to integer */

      intval = atoi(tmpstrng) ;

/*       Insert into format buffer */

      fmtbuffer += intval ;
   }
   return(fmtbuffer) ;
}
```

```
void    INIT_X25_LISTEN()
{

/* This routine sets up a listen on each of the X.25 links */
char    *bufptr ;
int     ii ;
ncb_t   far *intptr ;


/*  Loop for each of the translation table entries */


   for(ii=0 ; ii < MAX_TBL ; ii++)
   {

/*      If the currert entry is a non-local request or it has a TCP */
/*          address conversion specified for it */
/*      Then */

     if ((strcmp(trans_tbl[ii].isdnaddr,loc_isdn_addr) != 0 ||
          trans_tbl[ii].tcpaddr[0] != '\0') &&
          trans_tbl[ii].x25addr[0] != '\0')
     {

/*           Allocate the memory for the listen request */

        trans_tbl[ii].x25ptr = (ncb_t *) malloc(sizeof(ncb_t)) ;
        bufptr = malloc(128) ;

/*           Set up the listen for this address */

        x25_listen(trans_tbl[ii].x25ptr,trans_tbl[ii].x25addr,bufptr)
;
     }
   }
}
```

```
/*******************************************************/
/*** The following fole contains the code and file***/
/*** #includes which make up the Gateway State    ***/
/*** handling module.                             ***/
/*** This file is initcod.c                       ***/
/*******************************************************/


#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <graph.h>
#include <math.h>

/*  FRONTIER/TCP DECLARATIONS */

#include "c:\tcp\include\ftctypes.h"
#include "c:\tcp\include\ulpuser.h"
#include "c:\tcp\include\ulpshare.h"


/*  GATEWAY CONSTANT DECLARATIONS */

#include "c:\devel\gatedef.h"

/*  GATEWAY TYPE DECLARTATIONS */

#include "c:\devel\tipedef.h"

/*  EXTERNAL VARIABLE DECLARATIONS */

#include "c:\devel\globaldef.h"

/*  EXTERNAL SUBROUTINE DECLARATIONS */

#include "c:\devel\subdecl.h"

/*
   Setup a LISTEN on the X25 line...
*/
void x25_listen(ncb_t far *xncbp,char *addrptr, char  *bufptr)
{
   short   i,ii ;
   ushort   intval,tmpval ;
   uchar    tmpchar,*tmpchptr ;


   clear_ncb(xncbp);
   xncbp->ncb_command = 0x91; /*NABIOS_CALL;*/
```

```
   destp = xncbp->ncb_callname;
   srcp = X25_NAME;
   for (i=0; i<16; i++, srcp++, destp++)
      *destp = *srcp;
   xncbp->ncb_buffer = bufptr ;
   xncbp->ncb_lana_num = 0x0ff;

   rtn_ptr = x25_listen_post;
   xncbp->ui2 = FP_SEG(rtn_ptr);
   xncbp->ui1 = FP_OFF(rtn_ptr);

   xncbp->ncb_rto = 10;
   xncbp->ncb_sto = 10;

/*  Convert the string X.25 address to transmittable value */

   intval = strlen(addrptr) ;
   intval = intval << 4 ;
   call_buffer[0] = intval ;
   tmpchptr = addrptr ;

/*  Convert internal X.25 addresses to tx format */

   for(i=0,ii=1; i < strlen(addrptr) ; i += 2, ii++ )
   {

      intval = 0 ;
      tmpchar = *tmpchptr++ ;
      intval = tmpchar - 48 ;
      tmpchar = *tmpchptr++ ;
      intval = intval << 4 ;
      tmpval = tmpchar - 48 ;
      if (tmpval >= 0 )
      {
         intval += tmpval ;
      }
      call_buffer[ii] = intval ;

   }

   memcpy(&call_buffer[ii],perm_call_values,5) ;
   xncbp->ncb_length = ii + 5 ;
   memcpy(bufptr,call_buffer,xncbp->ncb_length) ;
   xncbp->ncb_retcode = 99 ;

   if (gattrace==1 || gattrace == 3)
   {
      sprintf(obuf,"Starting X.25 listen for %s",addrptr);
      output_message(obuf);
   }
```

```
   send_cmd_5c(xncbp);
}

void isdn_x25d()


/*
   Call to *X25D to receive the lsn.
*/
{
   short i;
   ncb_t far *incbp;
   ncb_t ncb;

   incbp = &ncb;
   PERF_COM_NCB_ACT(incbp) ;
   incbp->ncb_lsn = 0 ;
   incbp->ncb_command = 0x10;
   send_cmd_60(incbp);
   x25d_lsn = incbp->ncb_lsn;

}

void isdn_sip()
/*
   Call to *SIP to receive its lsn.
*/
{
   short i;
   ncb_t far *incbp;
   ncb_t ncb;
   struct sip_setup
   {
      short sip_command;
      short reserved;
      short call_type;
      short r_adapt;
      short l2_spec;
      short l3_spec;
      short l2_comp;
      short l3_comp;
      short channel;
      short lsn;
      short dial_length;
      char  dial[4];
   }ss;

   struct sip_response
```

```
   {
      short sip_command;
      short reserved;
      short lsn;
      short return_code;
      short previous_command;
   }sr;

   incbp = &ncb;
   clear_ncb(incbp);
   incbp->ncb_command = 0x10;
   destp = incbp->ncb_callname;
   srcp = SIP_NAME;
   for(i=0;i<NB_NAME_LEN;i++,srcp++,destp++)
      *destp = *srcp;
   send_cmd_60(incbp);
   sip_lsn = incbp->ncb_lsn;
/*
   Call to *SIP to send setup info.
*/
   incbp->ncb_command = 0x14;
   incbp->ncb_lsn = sip_lsn;
   incbp->ncb_buffer = (char *) &ss;
   incbp->ncb_length = sizeof(ss);
   ss.sip_command = 0x01;
   ss.reserved = 0x0;
   ss.call_type = 0x01;
   ss.r_adapt = 0x00;
   ss.l2_spec = 0x06;
   ss.l3_spec = 0x06;
   ss.l2_comp = 0x06;
   ss.l3_comp = 0x06;
   ss.channel = 0x00;
   ss.lsn = (short) x25d_lsn;
   ss.dial_length = 0x0;
   send_cmd_60(incbp);
   if (gattrace==1 || gattrace == 3)
   {
      sprintf(obuf,"SIP send setup return, dial length : %d\n",
                  ss.dial_length);
      output_message(obuf);
   }

/*
   Receive response from *SIP.
*/
   incbp->ncb_command = 0x15;
   incbp->ncb_lsn = sip_lsn;
   incbp->ncb_buffer = (char *) &sr;
   incbp->ncb_length = sizeof(sr);
```

```
   send_cmd_60(incbp);
/*
   Hangup to *SIP.
*/
   incbp->ncb_command = 0x12;
   incbp->ncb_lsn = sip_lsn;
   send_cmd_60(incbp);
}
void isdn_start_dpp()
/*
   Send START_DPP to *X25D.
*/
{
   short i;
   ncb_t far *incbp;
   ncb_t ncb;
   IOTCL_TYPE iotcl ;

   incbp = &ncb;
   PERF_COM_NCB_ACT(incbp) ;
   incbp->ncb_command = 0x14;   /* Send - 94 = no wait mode */
   incbp->ncb_lsn = x25d_lsn;
   incbp->ncb_buffer = (char *) &iotcl;
   incbp->ncb_length = sizeof(iotcl);

   iotcl.iotcl_cmd = 0x08;
   iotcl.iotcl_cmd_cmd = 0x01;
   iotcl.iotcl_cmd_arg[0] = 8;
   iotcl.iotcl_cmd_arg[1] = 0;
   iotcl.iotcl_cmd_arg[2] = 4;
   for(i=3;i<13;i++)
      iotcl.iotcl_cmd_arg[i] = 0;
   dpp_response = 0;
   i = 0;
   send_cmd_60(incbp);

   if (gattrace == 1 || gattrace == 3)
   {
      sprintf(obuf,"Layers 2 and 3 are up...\n");
      output_message(obuf);
   }
}

void shutdown()
{
/*
This routine stops the gateway processing and shuts down the I/O
*/

   int ii ;
```

```
/*
   If the gateway is currently up
   Then
*/
   if(gateway_up==1)
   {
/*
      Loop through the Translation Table
*/
      for(ii=0 ; ii < MAX_TBL; ii++)
      {
/*
         if there is a listen buffer allocated for the current
            TCP link
         Then
            free the buffer
*/
         if (trans_tbl[ii].tcpptr != 0)
         {
            free((char *) trans_tbl[ii].tcpptr) ;
         }
/*
         if there is a listen buffer allocated for the current
         Then
            Deallocate the listen buffer and the call buffer
*/

         if (trans_tbl[ii].x25ptr != 0)
         {
            free((char *) trans_tbl[ii].x25ptr->ncb_buffer) ;
            free((char *) trans_tbl[ii].x25ptr) ;
         }
      }

/*
      If the ISDN link has been initialized
      Then
         Hangup the ISDN logical session
*/

      if(menu_choice[1] == 1)
      {
         isdn_call_sw = 0;
         x25d_hangup();
      }
/*
      If the X.25 link has been initialized
      Then
```

```
        Hangup each open X.25 session
*/
     if(menu_choice[0] == 1)
     {
        x25_call_sw = 0;
        for(ii=0 ; ii < MAX_X25_OUT ; ii++)
        {
           if (x25_out[ii].int_call_typ != NOT_CONNECTED)
           {
              x25_hangup(x25_out[ii].cur_typ_id);
           }
        }
     }
/*
     If the TCP link has been initialized
     Then
        Hangup each open TCP session
*/
     if(menu_choice[2] == 1)
     {
        for(ii=0 ; ii < MAX_TCP_OUT ; ii++)
        {
           if (tcp_out[ii].int_call_typ != NOT_CONNECTED)
           {
              HANGUP_TCP_DEST(tcp_out[ii].cur_typ_id);
           }
        }
     }
   }
   hangup = 1;
   gateway_up = 0 ;
   sprintf(obuf,"Gateway is shutdown.\n");
   output_message(obuf);
}


/*
   This routine initates the TCP card load operation
*/
#include <c:\devel\in_tc_li.c>
/*
   This routine oversees the gateway startup process
*/
#include <c:\devel\set_gate.c>
/*
   This routine determines which TCP addresses from the
   Translation table require listens
*/
#include <c:\devel\in_tc_ls.c>
/*
```

```
   This routine initiates a TCP listen on a specified address/port
*/
#include <c:\devel\tcplisn.c>
/*
   This routine converts TCP addresses to the format required for a
listen
*/
#include <c:\devel\co_tc_tx.c>
/*
   This routine determines which X.25 addresses require listens
*/
#include <c:\devel\in_x_lis.c>
```

```
void    CHK_NET()
{
/*  This routine checks for any network activity which is required */

   UCMD locpcb ;


/*  If an X.25 listen has completed */
/*  Then */
/*       Queue the X.25 listen */

   if (xlisten == 1)
   {
      output_message("received listen") ;
      xlisten = 0 ;
      QUE_X25_LISN() ;
   }

/*  If an X.25 receiv has completed */
/*  Then */
/*       Queue the X.25 receive */

   if (x25_call_sw==1)
   {
      x25_call_sw = 0 ;
      QUE_X25_RX() ;
   }



/*  Queue up any pending ISDN actions */

   QUE_ISDN_ACT() ;

/*  Queue up any pending TCP actions */

   QUE_TCP_ACT(&locpcb) ;

/*  If there are any queued actions */
/*  Then */
/*       Choose action based on link type */

   if (bkptr != frptr)
   {
      switch(pend_act[bkptr].typ)
      {

/*          Case X.25 link */
/*              Determine and perform X.25 action to perform */
```

```
            case X25TYP :
               DET_X25_ACT(bkptr) ;
               break ;

/*               Case TCP link */
/*                   Determine and perform required TCP action */

            case TCPTYP :
               DET_TCP_ACT(bkptr) ;
               break ;

/*               Case ISDN link */
/*                   Determine and perform isdn action */

            case ISDNTYP :
               DET_ISDN_ACT(bkptr) ;
               break ;
         }
         if(++bkptr == MAX_PEND_ACT)
         {
            bkptr = 0 ;
         }
      }
   }
```

```
void    DET_X25_ACT(ushort qptr)
{
/*  This routine determines and performs the pending x.25 actions */

   struct timeb timebuffer ;
   long tmp_pack_time ;

/*  determine action based on pending action queue action */

   ftime(&timebuffer) ;

   switch(pend_act[qptr].act)
   {

/*      case listen completion indicated */
/*          Process the received incall message */

      case OP_LISTEN :
        tot_calls++ ;
        call_queued_time += ((timebuffer.time -
                        pend_act[qptr].qtime)*1000)
                     + (((long) timebuffer.millitm) -
                           pend_act[qptr].msec) ;
        pend_act[qptr].qtime = timebuffer.time ;
        pend_act[qptr].msec = timebuffer.millitm ;
        PROC_X25_INCALL(trans_tbl[pend_act[qptr].indx].x25ptr,
                pend_act[qptr].indx) ;
        ftime(&timebuffer) ;
        call_process_time += ((timebuffer.time - pend_act[qptr].qtime)
                        *1000)
                     + (((long) timebuffer.millitm) -
                          pend_act[qptr].msec) ;
        break ;

/*      case Data received on the X.25 link */
/*          Process the received data */

      case OP_RX :
        tot_packs++ ;
        if (ixncb[pend_act[qptr].indx].ncb_length > 0)
        {
          pack_queued_time += ((timebuffer.time -
pend_act[qptr].qtime)
                           *1000)
                     + (((long) timebuffer.millitm)
                        - pend_act[qptr].msec) ;
        }
        pend_act[qptr].qtime = timebuffer.time ;
        pend_act[qptr].msec = timebuffer.millitm ;
        PROC_RX_X25_DAT(pend_act[qptr].indx) ;
```

```
            ftime(&timebuffer) ;
            if (ixncb[pend_act[qptr].indx].ncb_length > 0)
            {
              pack_process_time += ((timebuffer.time -
                                pend_act[qptr].qtime)*1000
                          + (((long) timebuffer.millitm) -
                                pend_act[qptr].msec)) ;
            }
            else
              tot_packs-- ;

            break ;

/*          case End of data transmit */
/*              process end of data transmit received */

         case OP_TX :
            PROC_TX_X25_CONC(pend_act[qptr].indx) ;
            break ;
      }
   }
```

```
void    PROC_X25_INCALL(ncb_t *in_call_ptr,ushort tbl_ptr)
{
/*  This routine processes a received call indication from the X.25
link */

   ushort  source_typ,ii,success ;
   uchar   ll,rl ;
   char    *input_data,*curchar ;
   int     fnd_row ;

/*  Indicate X.25 call received */
/*  Unpack the x.25 addresses */

   success = 0 ;
   source_typ = X25TYP ;


   input_data,curchar = (char *) in_call_ptr->ncb_buffer ;

   ll = *curchar ;
   ll = (ll << 4) ;
   ll = ll >> 4 ;
   rl = *curchar++ ;
   rl = rl >> 4 ;

   remote_length = ll ;


   set_x25_address(ll,x25_send_address,curchar) ;
   curchar += (ll+1)/2 ;
   set_x25_address(rl,x25_recv_address,curchar) ;

/*  Choose out_call row to use for this call */
/*  if row was found */
/*  then */

   fnd_row = -1 ;
   for(ii=0; ii < MAX_X25_OUT && fnd_row == -1 ; ii++)
   {
      if (x25_out[ii].int_call_typ == NOT_CONNECTED)
      {
         fnd_row = ii ;
      }
   }

   if (fnd_row != -1)
   {

/*       Save the known information in the out_call buffer */
/*       Determine the destination of the call */
```

```
     x25_out[fnd_row].cur_typ_id = in_call_ptr->ncb_lsn ;
     x25_out[fnd_row].tar_typ_entry = tbl_ptr ;

     success = DET_X25_DEST(x25_send_address,x25_recv_address,
                         &x25_out[fnd_row]) ;

/*       If destination found and allowable */
/*       Then */

     if (success != 0)
     {

/*          Initiate the call to the destination */

        success = INIT_CALL_DEST(&x25_out[fnd_row],source_typ) ;

/*          If the call was initiated successfully */
/*          then */
/*              Accept the incoming call */

        if (success != 0)
        {
           ovr_stats.rem1_call_count++ ;
           x25_call_setup(fnd_row) ;

/*               Wait for incoming messages */

           x25_call_wait(fnd_row) ;

           if (gattrace == 1 || gattrace == 3)
           {
              sprintf(obuf,"Received call for %s",
                     trans_tbl[tbl_ptr].x25addr) ;
              output_message(obuf) ;

              if (x25_out[fnd_row].tar_typ == TCPTYP)
              {
                 sprintf(obuf,"Translated to TCP address: %s",
                   trans_tbl[x25_out[fnd_row].tar_typ_entry].tcpaddr) ;
                 output_message(obuf) ;
              }
              sprintf(obuf,"Received message from: %s",
                 trans_tbl[x25_out[fnd_row].cur_typ_entry].x25addr) ;
              output_message(obuf) ;
           }


        }
        else
```

```
        {
           x25_out[fnd_row].int_call_typ = NOT_CONNECTED ;
        }
      }
      else
      {
         x25_out[fnd_row].int_call_typ = NOT_CONNECTED ;
      }
   }
   else
   {
      output_message("Exceeded maximum number of calls on the X.25
link") ;
   }


/*  If the call setup was not successful */
/*  Then */
/*       Clear the incoming call */

   if (success == 0)
   {
      x25_hangup(in_call_ptr->ncb_lsn) ;


   }

/*  Queue a new listen for this address */

   x25_listen(in_call_ptr,trans_tbl[tbl_ptr].x25addr,
                                 in_call_ptr->ncb_buffer) ;

}
```

```
ushort  DET_X25_DEST( char *tx_addr , char *rx_addr , OUT_CALL *curout
)
{

/*  This routine determines the destination information for an
incoming */
/*      X.25 sourced call */

   int     ii,fnd_row ;

/*  Search the translation table for the destination address */

   fnd_row = 0 ;
   for( ii = 0 ; ii < MAX_TBL &&  fnd_row == 0 ; ii++)
   {
      if (strcmp(trans_tbl[ii].x25addr,rx_addr) == 0)
      {
         fnd_row = 1 ;
         curout->cur_typ_entry = ii ;
      }
   }

/*  If an entry was found */
/*  Then */
/*      Store the row in the given out_call row */
/*      If the target row has a tcp address entry */
/*      Then */
/*          Indicate target type is to be TCP */

   if (fnd_row != 0)
   {
      if (trans_tbl[curout->tar_typ_entry].tcpaddr[0] != '\0' &&
         trans_tbl[curout->cur_typ_entry].tcpaddr[0] != '\0')
      {
         curout->tar_typ = TCPTYP ;
      }

/*      Else */
/*          Indicate target type is X.25 */

      else
      {
         curout->tar_typ = X25TYP ;
      }
   }
/*  else */
/*      indicate request should be rejected */

/*  Check for duplicate call */
/*  If duplicate call found */
```

```
/*  Then */
/*       Indicate current call should be rejected */

   for(ii=0; ii<MAX_X25_OUT & fnd_row != 0 ; ii++)
   {
      if (x25_out[ii].int_call_typ != NOT_CONNECTED &&
         (curout->tar_typ_entry == x25_out[ii].tar_typ_entry &&
          curout->cur_typ_entry == x25_out[ii].cur_typ_entry))
      {
         fnd_row = 0 ;
      }
   }

   return(fnd_row) ;
}
```

```
ushort INIT_CALL_DEST(OUT_CALL *curout, ushort source_typ)
{
/* This routine sets up an outgoing call to a given link from a given
type */

   ushort success ;

/*  If destination ISDN is different from the local isdn value */
/*  Then */
/*       Set internal call type to ISDN */
/*  Else */
/*       Set internal call type to the target type */

   success = 0 ;
   if (strcmp(trans_tbl[curout->tar_typ_entry].isdnaddr,loc_isdn_addr)
!= 0)
   {
      curout->int_call_typ = ISDNTYP ;
   }
   else
   {
      curout->int_call_typ = curout->tar_typ ;
   }

/*  Determine actions based on internal call type */
   if ((curout->int_call_typ == X25TYP && menu_choice[0] == 1) ||
      (curout->int_call_typ == ISDNTYP && menu_choice[1] == 1) ||
      (curout->int_call_typ == TCPTYP && menu_choice[2] == 1))
   {
     switch(curout->int_call_typ)
     {

/*       Case X.25 destination */
/*           Initiate X.25 call */
/*           Update X.25 statistics if call successful */

      case X25TYP :
         success = INIT_X25_CALL(curout,source_typ) ;
         if (success != 0)
         {
            brk_stats[source_typ].rem1_call_count++ ;
            if (source_typ == TCPTYP)
            {
               brk_stats[X25TYP].rem1_call_count++ ;
            }
            else
            {
               brk_stats[X25TYP].rem2_call_count++ ;
            }
            ovr_stats.rem1_call_count++ ;
```

```
         }
         break ;

/*       Case TCP destination */
/*           Initiate the call on a TCP link */
/*           Update statistics if successful */

      case TCPTYP :
         success = INIT_TCP_CALL(curout,source_typ) ;
         if (success != 0)
         {
            if (source_typ == X25TYP)
            {
               brk_stats[source_typ].rem1_call_count++ ;
               brk_stats[TCPTYP].rem1_call_count++ ;
            }
            else
            {
               brk_stats[source_typ].rem2_call_count++ ;
               brk_stats[TCPTYP].rem2_call_count++ ;
            }
            tcp_call_count++ ;
         }
         break ;

/*       Case ISDN destination */
/*           Initiate the call on the ISDN */
/*           Update the statistics if successful */

      case ISDNTYP :
         success = INIT_ISDN_CALL(curout,source_typ) ;
         if (success != 0)
         {
            brk_stats[source_typ].rem2_call_count++ ;
            ovr_stats.rem2_call_count++ ;
         }
         break ;
      }

   }
   else
   {
      if (gattrace != 0)
      {
         success = 1 ;
         curout->int_call_typ = DISKTYP ;
      }
   }
```

```
    return(success) ;
}
```

```
void    PERF_COM_NCB_ACT(ncb_t far *incbp)
{
/*  This routine performs the common NCB actions required prior to use
*/

   int i ;

   clear_ncb(incbp);
   destp = incbp->ncb_callname;
   srcp = X25D_NAME;
   for(i=0;i<NB_NAME_LEN;i++,srcp++,destp++)
      *destp = *srcp;
   incbp->ncb_lsn = x25d_lsn;
}
```

```
ushort  INIT_ISDN_CALL(OUT_CALL *curout,ushort source_typ)
{

/*  This routine initiates an ISDN call upon notification such is
required */

   int     loc_lid,ii,fnd_row ;
   ushort  success ;

/*  Find an available OUT_CALL row */

   success = 0 ;
   fnd_row = -1 ;

   for(ii=0;ii<MAX_ISDN_OUT && fnd_row == -1; ii++)
   {
      if (isdn_out[ii].int_call_typ == NOT_CONNECTED)
      {
         fnd_row = ii ;
      }
   }

/*  If a row is available */
/*  Then */
/*      Open the d channel for this call */

   if (fnd_row != -1)
   {

      loc_lid=isdn_d_link_open(curout,source_typ) ;


/*      If the open succeeded */
/*      Then */
/*          Setup the transmit ncb for this link */

      if (loc_lid != -1)
      {
         success = 1 ;
         if (gattrace == 1 || gattrace == 3)
         {
            sprintf(obuf,"ISDN link opened successfully, link # %d",
                     loc_lid) ;
            output_message(obuf) ;
         }
         isdn_call_setup(loc_lid,&oincb[fnd_row],MAX_X25_OUT+fnd_row) ;

/*              Fill out the out_call row for this connection */

         curout->tar_typ_id = loc_lid ;
```

```
            isdn_out[fnd_row].cur_typ_entry = curout->tar_typ_entry ;
            isdn_out[fnd_row].int_call_typ = source_typ ;
            isdn_out[fnd_row].tar_typ = source_typ ;
            isdn_out[fnd_row].tar_typ_entry = curout->cur_typ_entry ;
            isdn_out[fnd_row].tar_typ_id = curout->cur_typ_id ;
            isdn_out[fnd_row].cur_typ_id = loc_lid ;

/*                  Format and send the isdn header message */

            FMT_ISDN_HDR(curout,source_typ,fnd_row) ;

/*                  Update isdn statistics for this call */

            if (source_typ == X25TYP)
            {
              brk_stats[ISDNTYP].rem1_call_count++ ;
            }
            else
            {
              brk_stats[ISDNTYP].rem2_call_count++ ;
            }

        }
    }
    return(success) ;
}
```

```
void    DISC_ISDN_LINK(ushort lid)
{
/*  This routine disconnects a specified link on the ISDN */

   struct {
      short   disc_c_cmd ;
      short   disc_c_lid ;
      short   disc_c_cause ;
      short   disc_c_diag ;
   }loc_disc ;

   ncb_t locncb ;
   ncb_t far *ncpptr ;

/*  Setup the ncb */

   ncpptr = &locncb ;
   PERF_COM_NCB_ACT(ncpptr) ;

   locncb.ncb_command = 0x14 ;
   locncb.ncb_buffer = (char *) &loc_disc ;
   locncb.ncb_length = sizeof(loc_disc) ;

/*  Setup the disconnect structure */

   loc_disc.disc_c_lid = lid ;
   loc_disc.disc_c_cmd = 0x07 ;
   loc_disc.disc_c_cause = 0 ;
   loc_disc.disc_c_diag = 0 ;

/*  Send the command to the card */

   send_cmd_60(ncpptr) ;

}
```

```
char    destyp[] = "TX" ;

void    FMT_ISDN_HDR(OUT_CALL *curout,ushort source_typ,ushort currow)
{

/*  This routine formats and issues an ISDN header message */

   char    fmt_buffer[X25_ADDR_LEN+X25_ADDR_LEN+6] ;
   int     tmp_len,lenvrbl ;

/*  Set destination type into buffer */

   tmp_len = 0 ;
   fmt_buffer[0] = '\0' ;

   sprintf(fmt_buffer,"%s%c",fmt_buffer,
            *(destyp+curout->tar_typ)) ;
   tmp_len++ ;

/*  Set destination and Source address into buffer */

   lenvrbl = X25_ADDR_LEN ;

   if (curout->tar_typ == X25TYP)
   {
      sprintf(fmt_buffer,"%s%*s  %*s  ",fmt_buffer,lenvrbl,
             trans_tbl[curout->tar_typ_entry].x25addr,lenvrbl,
               trans_tbl[curout->cur_typ_entry].x25addr) ;
   }
   else
   {
      sprintf(fmt_buffer,"%s%*s%2d%*s%2d",fmt_buffer,lenvrbl,
             trans_tbl[curout->tar_typ_entry].tcpaddr,
             trans_tbl[curout->tar_typ_entry].tcpport,lenvrbl,
             trans_tbl[curout->cur_typ_entry].tcpaddr,
             trans_tbl[curout->cur_typ_entry].tcpport) ;
   }

      tmp_len += 2*X25_ADDR_LEN + 4 ;

/*  Transmit the message on the ISDN link */

   ou[MAX_X25_OUT+currow].dm.data_Size = tmp_len ;
   ovr_stats.rem2_packets_sent++ ;
   ovr_stats.rem2_bytes_sent += tmp_len ;

   memcpy(ou[MAX_X25_OUT+currow].dm.data_Data,fmt_buffer,tmp_len) ;

   if (gattrace == 1 || gattrace == 3)
   {
```

```
        sprintf(obuf,"Formatted header: %s",
          ou[MAX_X25_OUT+currow].dm.data_Data);
        output_message(obuf) ;
      }

      send_cmd_60(&oincb[currow]) ;

    }
```

```
void    QUE_TCP_ACT(UCMD *tmpucmd)
{
/*  This routine requests any pending TCP actions and queues them for
*/
/*      processing by the gateway */

   SHCMD   loc_sh_cmd ;
   int     tmp_stat ;
   UCMD    loc_ucmd ;
   struct  timeb timebuffer ;

/*  Call the check for required actions routine */
/*  If there is a pending TCP action */
/*  Then */

   if (ulprxcmd(&loc_sh_cmd,&loc_ucmd,data_buffer) == PROTO_TCP)
   {

      if (gattrace == 1 || gattrace == 3)
      {
         sprintf(obuf,"Completed TCP action: shcmd: %d , pcbcmd:%d\n",
                 loc_sh_cmd.command,loc_ucmd.tcp_pcb.p_op) ;
         output_message(obuf) ;
         sprintf(obuf,"Status return code was: %d\n",
                          loc_ucmd.tcp_pcb.p_stat) ;
         output_message(obuf) ;
      }


/*      Store action, link type and process id in the action queue */

      if (loc_sh_cmd.command != 1)
      {
         if (loc_ucmd.tcp_bcb.b_res == 0 &&
              (loc_ucmd.tcp_bcb.b_flags == STAT_REMCLS ||
               loc_ucmd.tcp_bcb.b_flags == STAT_FLUSHED ||
               loc_ucmd.tcp_bcb.b_flags == STAT_RESET))
         {
            loc_sh_cmd.command = OP_CLOSE ;
            loc_ucmd.tcp_pcb.p_stat = STAT_REMCLS ;
         }
         else
         {
            loc_sh_cmd.command = loc_sh_cmd.command + OP_STATUS ;
         }
      }
      else
      {
         loc_sh_cmd.command = loc_ucmd.tcp_pcb.p_op ;
      }
```

```
      pend_act[frptr].typ = TCPTYP ;
      pend_act[frptr].act = loc_sh_cmd.command ;
      pend_act[frptr].indx = loc_sh_cmd.pid ;
      ftime(&timebuffer) ;
      pend_act[frptr].qtime = timebuffer.time ;
      pend_act[frptr].msec = timebuffer.millitm ;

/*       Determine further actions based on command */

      switch(loc_sh_cmd.command)
      {

/*           Case received data command */
/*               Copy the BCB into its specified buffer */
/*               Copy the data into its specified buffer */

         case OP_RX :
            memcpy(&globucmd[loc_sh_cmd.pid],&loc_ucmd,sizeof(BCB)) ;
            memcpy(tcpbuf[loc_sh_cmd.pid],data_buffer,
                   loc_ucmd.tcp_bcb.b_res) ;
            if (++frptr == MAX_PEND_ACT)
            {
               frptr = 0 ;
            }
            break ;

/*           Case Completed data transmission */
/*               copy the BCB to its specific buffer */

         case OP_TX :
            memcpy(&globucmd[loc_sh_cmd.pid+MAX_TCP_OUT],&loc_ucmd,
                                    sizeof(BCB)) ;
            if (++frptr == MAX_PEND_ACT)
            {
               frptr = 0 ;
            }
            break ;

/*           Case Connect command completion */
/*               Indicate connect command has completed */

         case OP_CONNECT :
            memcpy(tmpucmd,&loc_ucmd,sizeof(PCB)) ;
            tcp_call_connected = 1 ;
            break ;

/*           Case statistics command completed */
/*               Indicate statistics command has completed */
```

```
        case OP_STATUS :
           memcpy(tmpucmd,&loc_ucmd,sizeof(PCB)) ;
           tcp_stats_rx = 1 ;
           break ;

/*           Case Listen command completed */
/*               Copy the PCB to the appropriate buffer */

        case OP_LISTEN :

memcpy(trans_tbl[loc_sh_cmd.pid].tcpptr,&loc_ucmd,sizeof(PCB)) ;
           if (++frptr == MAX_PEND_ACT)
           {
              frptr = 0 ;
           }
           break ;

/*           Case Close command */
/*               Copy the PCB to the appropriate buffer */


        case OP_CLOSE :
           memcpy(&globucmd[loc_sh_cmd.pid],&loc_ucmd,
                                              sizeof(PCB)) ;
           if (++frptr == MAX_PEND_ACT)
           {
              frptr = 0 ;
           }
           break ;
     }
   }
}
```

```
ushort  INIT_TCP_CALL(OUT_CALL *curout,ushort source_typ)
{

/*  This routine initiates a call on the TCP link */

   PCB     conpcb ;
   BCB     *curbcb ;
   PCB     *pcbptr ;
   int     ii,fnd_row,cur_stat ;
   ushort  success ;

/*  Format the current PCB */


   pcbptr = &conpcb ;
   success = 0 ;

   conpcb.p_op = OP_CONNECT ;
   conpcb.p_id = 0 ;
   conpcb.p_protocol = PROTO_TCP ;
   conpcb.p_lport = trans_tbl[curout->cur_typ_entry].tcpport ;
   conpcb.p_fport = trans_tbl[curout->tar_typ_entry].tcpport ;
   conpcb.p_lhost = 0 ;
   conpcb.p_fhost =
CONV_TCP_TX(trans_tbl[curout->tar_typ_entry].tcpaddr) ;
   conpcb.p_stat = 0 ;
   conpcb.p_ttl = 0 ;
   conpcb.p_tos = 0 ;
   conpcb.p_tcc = 0 ;
   conpcb.p_retrantimeout = 0 ;

/*  Find a TCP out_call row */

   fnd_row = -1 ;
   for(ii=0 ; ii < MAX_TCP_OUT && fnd_row == -1 ; ii++)
   {
      if (tcp_out[ii].int_call_typ == NOT_CONNECTED)
      {
         fnd_row = ii ;
      }
   }
   conpcb.p_pid = fnd_row ;

/*  Issue the connect command */

   cur_stat = TCPCTL(pcbptr) ;
```

```
/*  If the command has issued correctly */
/*  Then */

   if (cur_stat == 0)
   {

/*      Wait for the operation to complete */


      tcp_call_connected = 0 ;
      while (tcp_call_connected == 0)
      {
         QUE_TCP_ACT((UCMD *) &conpcb) ;
      }

/*       if the connection succeeded */
/*       Then */
/*           Format TCP out call row */

      if (conpcb.p_stat == 0)
      {
         tcp_out[fnd_row].cur_typ_id = pcbptr->p_id ;
         curout->tar_typ_id = pcbptr->p_id ;
         tcp_out[fnd_row].cur_typ_entry = curout->tar_typ_entry ;
         tcp_out[fnd_row].tar_typ = source_typ ;
         tcp_out[fnd_row].int_call_typ = source_typ ;
         tcp_out[fnd_row].tar_typ_id = curout->cur_typ_id ;
         tcp_out[fnd_row].tar_typ_entry = curout->cur_typ_entry ;

         if (gattrace == 1 || gattrace == 3)
         {
            sprintf(obuf,"Initiated TCP call: id: %d, pid: %d",
                 pcbptr->p_id,fnd_row) ;
            output_message(obuf) ;
         }


/*           Queue a receive for this connection */

         curbcb = (BCB *) &globucmd[fnd_row] ;

         curbcb->b_id = pcbptr->p_id ;
         curbcb->b_cnt = 128 ;
         curbcb->b_pid = fnd_row ;
         curbcb->b_flags = 0 ;

         cur_stat = TCPRX(curbcb) ;

         success = 1 ;
```

```
/*          Fill out the transmit BCB structure */

        globucmd[MAX_TCP_OUT+fnd_row].tcp_bcb.b_id = pcbptr->p_id ;
        globucmd[MAX_TCP_OUT+fnd_row].tcp_bcb.b_pid = fnd_row ;
      }
   }
   return(success) ;
}
```

```
void    PROC_RX_X25_DAT(ushort currow)
{

/*  This routine processes data packets as they are received on the
x.25*/

   ushort  success ;

/*  If there is data in the current buffer */
/*  Then */
/*      Update the X.25 statistics accordingly */

   if (ixncb[currow].ncb_length > 0)
   {
      ovr_stats.rem1_packets_recv++ ;
      ovr_stats.rem1_bytes_recv += ixncb[currow].ncb_length ;
      memcpy(lpx.dm.data_Data,ixncb[currow].ncb_buffer,
                     ixncb[currow].ncb_length) ;

      lpx.dm.data_Size = ixncb[currow].ncb_length ;

      if (gattrace == 1 || gattrace == 3)
      {
         sprintf(obuf,"Received X.25 data: Link: %d, Size: %d",
                x25_out[currow].cur_typ_id,ixncb[currow].ncb_length) ;
         output_message(obuf) ;
      }

/*      Transmit the data to the chosen target */

      success = INIT_TX_DEST(&x25_out[currow],ixncb[currow].ncb_length,
                     ixncb[currow].ncb_buffer,X25TYP) ;

/*      If the transmission was successful */
/*      Then */
/*          Wait for additional data */

      if ((success != 0) && ((ixncb[currow].ncb_retcode==0x00) ||
                (ixncb[currow].ncb_retcode==0x06)))
      {
         x25_call_wait(currow) ;
      }
   }

/*  if a hangup is indicated or there has been some error in the
receive */
/*  Then */
/*      Hang up the destination */

   if ((ixncb[currow].ncb_retcode != 0x00 &&
```

```
        ixncb[currow].ncb_retcode != 0xff &&
          ixncb[currow].ncb_retcode != 0x06) ||
                        success == 0)
   {
      if (gattrace == 1 || gattrace == 3)
      {
         sprintf(obuf,"Received X.25 remote hangup or error,lid: %d",
                x25_out[currow].cur_typ_id) ;
         output_message(obuf) ;
      }


      HANGUP_DEST(&x25_out[currow]) ;

/*       Clear the out_call row of this call */

      x25_out[currow].int_call_typ = NOT_CONNECTED ;

/*       If the hangup is due to some error and not a request */
/*       Then */
/*           Hang up the X.25 destination */

      if (ixncb[currow].ncb_retcode != 0x0a)
      {
         sprintf(obuf,"Received X.25 NCB error retcode %d",
                        ixncb[currow].ncb_retcode) ;
         output_message(obuf) ;
         x25_out[currow].int_call_typ = X25TYP ;
         x25_out[currow].tar_typ_id = x25_out[currow].cur_typ_id ;
         HANGUP_DEST(&x25_out[currow]) ;
      }
   }
}
```

```
ushort  INIT_TX_DEST(OUT_CALL *curout,ushort size,char far * bufptr,
                                        ushort source_typ)
{

/* This routine initiates a transmit to the target when data has been
Rxd */

   ushort success ;

/*  Determine action based on maintained internal type */

   switch(curout->int_call_typ)
   {

/*       Case X.25 Destination */
/*           Transmit the message on the X.25 */
/*           Update source statistics accordingly */

      case X25TYP :
         success = TX_X25_MSG(curout,size,bufptr,source_typ) ;
         brk_stats[source_typ].rem1_packets_sent++ ;
         brk_stats[source_typ].rem1_bytes_sent += size ;
         break ;

/*       Case TCP destination */
/*           Transmit the message on the TCP link */
/*           Update the source type statistics accordingly */

      case TCPTYP :
         success = TX_TCP_MSG(curout,size,bufptr,source_typ) ;
         if (source_typ == X25TYP)
         {
            brk_stats[X25TYP].rem1_packets_sent++ ;
            brk_stats[X25TYP].rem1_bytes_sent += size ;
         }
         else
         {
            brk_stats[ISDNTYP].rem2_packets_sent++ ;
            brk_stats[ISDNTYP].rem2_bytes_sent += size ;
         }
         break ;

/*       case ISDN destination */
/*           Transmit the message on the ISDN link */
/*           Update the source type statistyics accordingly */

      case ISDNTYP :
         success = TX_ISDN_MSG(curout,size,bufptr,source_typ) ;
         brk_stats[source_typ].rem2_packets_sent++ ;
         brk_stats[source_typ].rem2_bytes_sent += size ;
```

```
            break ;

        }

        if (curout->int_call_typ == DISKTYP ||
            gattrace > 1)
        {
           success = 1 ;
           *(bufptr+size) = '\0' ;
           fprintf(logptr,"LID %d data %s\n",curout->cur_typ_id,bufptr) ;
        }
        return(success) ;
    }
```

```
void    HANGUP_DEST(OUT_CALL *curout)
{

/*  This routine hangs up the destination node */
   int     fnd_row ;
   ushort  ii ;


/*  Determine action based on internal call type */

   switch(curout->int_call_typ)
   {

/*      Case X.25 Destination */
/*          Find the current out_call row for this X.25 call */

     case X25TYP :
        fnd_row = -1 ;
        for(ii = 0 ; ii < MAX_X25_OUT && fnd_row == -1 ; ii++ )
        {
           if (x25_out[ii].int_call_typ != NOT_CONNECTED &&
                x25_out[ii].cur_typ_id ==
                          curout->tar_typ_id)
           {
              fnd_row = ii ;
           }
        }

/*           If the row was found */
/*           Then */

        if (fnd_row != -1)
        {

/*              Cancel any outstanding ncb's against this session */

           x25_cancel(x25_out[fnd_row].cur_typ_id,fnd_row) ;

/*              Hang up the link */

           x25_hangup(x25_out[fnd_row].cur_typ_id) ;

/*              Clear the call from the X.25 out_call structure */

           x25_out[fnd_row].int_call_typ = NOT_CONNECTED ;
        }
        break ;

/*      Case TCP Destination */
/*          Hang up the TCP link */
```

```
      case TCPTYP :
         HANGUP_TCP_DEST(curout->tar_typ_id) ;
         break ;
/*       Case ISDN Destination */
/*           Hang up the ISDN link */

      case ISDNTYP :
         HANGUP_ISDN_DEST(curout->tar_typ_id) ;
         break ;

   }
}
```

```
ushort  TX_ISDN_MSG(OUT_CALL *curout,ushort size,char far *bufptr,
                                          ushort source_typ)
{
/*  This routine transmits a received data message on the ISDN link */

   int     ii,fnd_row ;
   ushort  success ;

/*  Find the corresponding entry in the ISDN out_call structure */

   if (gattrace == 1 || gattrace == 3)
   {
      sprintf(obuf,"Initiating ISDN Transmit: id: %d size: %d, lsn:
%d",
                        curout->tar_typ_id,size,x25d_lsn ) ;
      output_message(obuf) ;
   }

   fnd_row = -1 ;
   for(ii=0; ii<MAX_ISDN_OUT && fnd_row == -1 ; ii++)
   {
      if (isdn_out[ii].int_call_typ != NOT_CONNECTED &&
           isdn_out[ii].cur_typ_id == curout->tar_typ_id)
      {
         fnd_row = ii ;
      }
   }

/*  Copy the data to the output buffer */

   isdn_call_setup(curout->tar_typ_id,&oincb[fnd_row],MAX_X25_OUT +
fnd_row) ;

   memcpy(ou[MAX_X25_OUT + fnd_row].dm.data_Data,bufptr,size) ;
   ou[MAX_X25_OUT + fnd_row].dm.data_Size = size ;

/*  Update the ISDN level statistics */

   if (source_typ == X25TYP)
   {
      brk_stats[ISDNTYP].rem1_packets_recv++ ;
      brk_stats[ISDNTYP].rem1_bytes_recv += size ;
   }
   else
   {
      brk_stats[ISDNTYP].rem2_packets_recv++ ;
      brk_stats[ISDNTYP].rem2_bytes_recv += size ;
   }

/*  Send the data on the isdn */
```

```
   send_cmd_60(&oincb[fnd_row]) ;

/*  If send was successful */
/*  Then */
/*       Update link level statistics */
/*       Indicate success */

   if (oincb[fnd_row].ncb_retcode == 0x00)
   {
      ovr_stats.rem2_packets_sent++ ;
      ovr_stats.rem2_bytes_sent += size ;
      success = 1 ;
   }
   else
   {
      sprintf(obuf,"ISDN data transmission error # %xd",
                        oincb[fnd_row].ncb_retcode) ;
      output_message(obuf) ;
   }
   return(success) ;
}
```

```
/*************************************************/
/*** The following code and #include files make***/
/*** up the Call Processing Module.            ***/
/*** This file is callproc.c                   ***/
/*************************************************/


#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <graph.h>
#include <math.h>

/*  FRONTIER/TCP DECLARATIONS */

#include "c:\tcp\include\ftctypes.h"
#include "c:\tcp\include\ulpuser.h"
#include "c:\tcp\include\ulpshare.h"

/*  GATEWAY CONSTANT DECLARATIONS */

#include "c:\devel\gatedef.h"

/*  GATEWAY TYPE DECLARATIONS */

#include "c:\devel\ipedef.h"

/*  EXTERNAL DATA DECLARATIONS */

#include "c:\devel\globaldef.h"

/*  EXTERNAL ROUTINE DECLARATIONS */

#include "c:\devel\subdecl.h"

void set_x25_address(uchar l, char *cp, char *ap)
{
/*
   This routine converts received X.25 addresses to internal
*/

   short i,j = 0;
   uchar t1,t2;
   char  *dstchr,*srcchr ;

   dstchr = cp ;
   srcchr = ap ;

   i = 0;
```

```
    j = 0;
    while(i<l/2)
    {
       t1 = *srcchr;
       t2 = *srcchr++;
       t1 = (t1>>4);
       t2 = (t2<<4);
       t2 = (t2>>4);
       sprintf(dstchr++,"%c",(t1+48));
       j++ ;
       if (j < l)
       {
          sprintf(dstchr++,"%c",(t2+48));
          j++ ;
       }
       i++;
    }
    sprintf(dstchr,"%c",'\0');

}

int isdn_d_link_open(OUT_CALL *curout,ushort source_typ)
/*
   Send a LINK_OPEN message to *X25D.
*/
{
   int i;
   short ad;
   short dl;
   ncb_t far *incbp;
   ncb_t ncb;
   struct link_struct
   {
      short link_op_cmd;
      short link_op_size;
      char link_op_control[64];
   }link;

   incbp = &ncb;
   PERF_COM_NCB_ACT(incbp) ;
   incbp->ncb_command = 0x14;
   incbp->ncb_buffer = (char *) &link;
   incbp->ncb_length = sizeof(link);
   link.link_op_cmd = 0x05;
   link.link_op_size = 64;
   for (i=0; i<64; i++)
      link.link_op_control[i] = 0x0;
/*
   Copy the addresses to the link open packet
*/
```

```
   i = strlen(trans_tbl[curout->tar_typ_entry].isdnaddr) ;
   memcpy(&link.link_op_control[14],
        trans_tbl[curout->tar_typ_entry].isdnaddr,i) ;
   link.link_op_control[i+14]=0x0;
   i = strlen(trans_tbl[curout->cur_typ_entry].isdnaddr) ;
   memcpy(&link.link_op_control[30],
        trans_tbl[curout->cur_typ_entry].isdnaddr,i) ;
   link.link_op_control[i+30]=0x0 ;
   link.link_op_control[0] = 0x0080 ;
   link.link_op_control[48] = 1 ;
   link.link_op_size = 51 ;
   link.link_op_control[46] = cur_sess++ ;

   if (gattrace==1 || gattrace == 3)
   {
      sprintf(obuf,"Calling ISDN address:%s, lsn: %d",
             &link.link_op_control[14],incbp->ncb_lsn);
      output_message(obuf);
   }

   send_cmd_60(incbp);

   if (gattrace == 1 || gattrace == 3)
   {
      sprintf(obuf,"ISDN call NCB return code: %x",incbp->ncb_retcode)
;
      output_message(obuf) ;
   }


/*
   Receive RESPONSE back from *X25D.
*/
   isdn_call_connected = 0 ;
   while (isdn_call_connected == 0)
   {
      kbhit() ;
      QUE_ISDN_ACT() ;
   }
   lid = r.link_op_LID;
   if (gattrace == 1 || gattrace == 3)
   {
      sprintf(obuf,"%s - ISDN connection established.",
                           prot_nam[source_typ]);
      output_message(obuf);
      sprintf(obuf,"Link identifier      : %d",r.link_op_LID);
      output_message(obuf) ;
      sprintf(obuf,"Session Number       : 0x%x",r.link_op_session);
      output_message(obuf) ;
   }
```

```
   if(r.link_op_ret_code == 0)
   {
      return(lid) ;
   }
   else
   {
      sprintf(obuf,"Isdn link open return code:
0x%x",r.link_op_ret_code);
      output_message(obuf) ;
      return(-1) ;
   }
}

void x25d_hangup()
/*
   Hangup the *X25D.
*/
{
   short i;
   ncb_t far *incbp;
   ncb_t ncb;

   incbp = &ncb;
   PERF_COM_NCB_ACT(incbp) ;
   incbp->ncb_command = 0x12;
   send_cmd_60(incbp);
   if (gattrace == 1 || gattrace == 3)
   {
      sprintf(obuf,"ISDN D channel session hangup.\n");
      output_message(obuf);
   }
}

void x25_hangup(ushort loc_lsn)
/*
   Hangup the X25 channel.
*/
{
   short i;
   ncb_t far *xncbp;
   ncb_t ncb;
   char    hang_buffer[256];

   xncbp = &ncb;
   clear_ncb(xncbp);
   destp = xncbp->ncb_callname;
   srcp = X25_NAME;
   for(i=0;i<NB_NAME_LEN;i++,srcp++,destp++)
      *destp = *srcp;
```

```
   xncbp->ncb_command = 0x12;
   xncbp->ncb_lana_num = 0x0ff;
   xncbp->ncb_lsn = loc_lsn ;
   xncbp->ncb_buffer = hang_buffer;
   xncbp->ncb_length = 0;
   send_cmd_5c(xncbp);
   if (gattrace == 1 || gattrace == 3)
   {
      sprintf(obuf,"X25 hangup on logical session # %d.",loc_lsn);
      output_message(obuf);
   }
}

void x25_cancel(ushort currow,ushort loc_lsn)
{

/*  This routine will cancel the outstanding listen for a given X.25
call */
   ncb_t far *cnp;
   ncb_t ncb;
   int i;

   clear_ncb(cnp);
   cnp = &ncb;
   cnp->ncb_command = 0x35;
   cnp->ncb_length = 0;
   cnp->ncb_buffer = (char *) &ixncb[currow];
   destp = cnp->ncb_callname;
   srcp = X25_NAME;
   for(i=0;i<NB_NAME_LEN;i++,srcp++,destp++)
       *destp = *srcp;
   cnp->ncb_lsn = loc_lsn;
   cnp->ncb_lana_num = 0xff;
   send_cmd_5c(cnp);
}

/*
   This routine processes received X.25 call requests
*/
#include <c:\devel\pr_x_incl.c>
/*
   This routine processes received TCP call requests
*/
#include <c:\devel\pr_t_call.c>
/*
   This routine processes received ISDN call requests
*/
#include <c:\devel\pr_i_call.c>
/*
   This routine determines the Destination information for X.25 calls
```

```
*/
#include <c:\devel\det_x_dst.c>
/*
   This routine determines the Destination information for TCP calls
*/
#include <c:\devel\det_t_dst.c>
/*
   This routine processes the received ISDN call header & opens the
channel
*/
#include <c:\devel\pr_i_hdr.c>
/*
   This routine converts TCP addresses to string from transmittable
format
*/
#include <c:\devel\con_t_rx.c>
/*
   This routine converts X.25 addresses from string to transmittable
format
*/
#include <c:\devel\con_x_tx.c>
/*
   This routine converts ISDN addresses to entry info
   from transmittable format
*/
#include <c:\devel\con_i_rx.c>
/*
   This routine oversees initiation the call to the destination link
*/
#include <c:\devel\in_ca_ds.c>
/*
   This routine opens a channel to a specified ISDN destination
*/
#include <c:\devel\in_is_call.c>
/*
   This routine opens a channel to a specified TCP destination
*/
#include <c:\devel\in_tc_call.c>
/*
   This routine opens a channel to a specified X.25 destination
*/
#include <c:\devel\in_x_call.c>
/*
   This routine formats & transmits an ISDN call header to the remote
gateway
*/
#include <c:\devel\fm_is_hdr.c>
/*
   This routine hangs up the destination channel for a given call
*/
```

```
#include <c:\devel\hangdst.c>
/*
  This routine processes the results of a TCP channel close
*/
#include <c:\devel\pr_tc_cls.c>
/*
  This routine processes a received TCP remote close indication
*/
#include <c:\devel\pr_i_rcls.c>
/*
  This routine closes a specified TCP destination link for a given
call
*/
#include <c:\devel\hng_t_dst.c>
/*
  These routines close a specified ISDN destination link for a given
call
*/
#include <c:\devel\hng_i_dst.c>
#include <c:\devel\ds_is_link.c>
```

```
ushort  TX_TCP_MSG(OUT_CALL *curout,ushort size,char far *bufptr,
                   ushort source_typ)
{
/*  This routine transmits a data buffer on the TCP link */

   int     ii,fnd_row,success ;

/*  Find the corresponding row in the out_call structure */

   if (gattrace == 1 || gattrace == 3)
   {
      sprintf(obuf,"Transmitting TCP packet on link # %d Size %d",
                  curout->tar_typ_id,size) ;
      output_message(obuf) ;
   }

   for(ii=0,fnd_row= -1; ii < MAX_TCP_OUT && fnd_row == -1 ; ii++)
   {
      if (tcp_out[ii].int_call_typ != NOT_CONNECTED &&
           tcp_out[ii].cur_typ_id == curout->tar_typ_id)
      {
         fnd_row = ii ;
      }
   }

/*  Fill out the bcb structure for this row */

   globucmd[MAX_TCP_OUT+fnd_row].tcp_bcb.b_cnt = size ;
   memcpy(data_buffer,bufptr,size) ;
   globucmd[MAX_TCP_OUT+fnd_row].tcp_bcb.b_flags = 0x0 ;

/*  Issue the command to the TCP processor */

   success = TCPTX(&globucmd[MAX_TCP_OUT+fnd_row],data_buffer) ;

/*  Update the protocol specific statisitcs */

   if (source_typ == X25TYP)
   {
      brk_stats[TCPTYP].rem1_packets_recv++ ;
      brk_stats[TCPTYP].rem1_bytes_recv += size ;
   }
   else
   {
      brk_stats[TCPTYP].rem2_packets_recv++ ;
      brk_stats[TCPTYP].rem2_bytes_recv += size ;
   }

/*  return success of the operation */
```

```
    if (success == 0)
    {
       return(1) ;
       sprintf(obuf,"Encountered TCP transmit failure %d",
                 globucmd[MAX_TCP_OUT + fnd_row].tcp_pcb.p_stat) ;
       output_message(obuf) ;
    }
    else
    {
       return(0) ;
    }
}
```

```
void    DET_TCP_ACT(ushort qptr)
{
/*  This routine determines the TCP action to be performed based on
the */
/*      completed action field of the pending action queue */

   struct timeb timebuffer ;

/*  Determine action based on stored action type */

   ftime(&timebuffer) ;

   switch(pend_act[qptr].act)
   {

/*      Case received listen response */
/*          Process TCP incoming call */

      case OP_LISTEN :
         tot_calls++ ;
         call_queued_time += ((timebuffer.time -
                     pend_act[qptr].qtime)*1000)
                     + (((long) timebuffer.millitm) -
                          pend_act[qptr].msec) ;
         pend_act[qptr].qtime = timebuffer.time ;
         pend_act[qptr].msec = timebuffer.millitm ;
         PROC_TCP_INCALL(pend_act[qptr].indx) ;
         ftime(&timebuffer) ;
         call_process_time += ((timebuffer.time -
                        pend_act[qptr].qtime)*1000)
                     + (((long)timebuffer.millitm) -
                        pend_act[qptr].msec) ;
         break ;

/*      Case received TCP data message */
/*          Process the incoming TCP data */

      case OP_RX :
         tot_packs++ ;
         pack_queued_time += ((timebuffer.time -
                        pend_act[qptr].qtime)*1000)
                     + (((long)timebuffer.millitm) -
                          pend_act[qptr].msec) ;
         pend_act[qptr].qtime = timebuffer.time ;
         pend_act[qptr].msec = timebuffer.millitm ;
         PROC_RX_TCP_DAT(pend_act[qptr].indx) ;
         ftime(&timebuffer) ;
         pack_process_time += ((timebuffer.time -
                        pend_act[qptr].qtime)*1000)
                     + (((long) timebuffer.millitm) -
```

```
                                pend_act[qptr].msec) ;
            break ;

/*       Case received TCP transmit concluded indication */
/*           Perform transmit concluded functions */

         case OP_TX :
            PROC_TX_TCP_CONC(pend_act[qptr].indx) ;
            break ;

/*       Case received TCP Close link message */
/*           Process Close link indication */

         case OP_CLOSE :
            PROC_TCP_CLOSE(pend_act[qptr].indx) ;
            break ;

      }
   }
```

```
void    PROC_TX_TCP_CONC(ushort indx)
{
/*  This routine processes the results of the concluded TCP transmit
*/

/*  If the transmission concluded successfully */
/*  Then */
/*       Update the link level TCP statistics */

   if (globucmd[indx+MAX_TCP_OUT].tcp_bcb.b_res != 0)
   {
      tcp_packets_sent++ ;
      tcp_bytes_sent += globucmd[indx+MAX_TCP_OUT].tcp_bcb.b_res ;
   }

/*  Else */
/*       Hang up the destination link */

   else
   {
/*       Indicate error to operator */


output_message(neterr(globucmd[indx+MAX_TCP_OUT].tcp_bcb.b_flags)) ;

      HANGUP_DEST(&tcp_out[indx]) ;

/*       Hang up the TCP link */

      HANGUP_TCP_DEST(tcp_out[indx].cur_typ_id) ;


   }
}
```

```
void    PROC_TCP_CLOSE(ushort lindx)
{
/*  This routine processes remote close indications and local close
results */

/*  If the current close is a remote close */
/*  Then */
/*       Hangup Destination link */

   if (globucmd[lindx].tcp_pcb.p_stat == STAT_REMCLS)
   {
      HANGUP_DEST(&tcp_out[lindx]) ;
   }

/*  Clear the connection from the internal table */

   if (lindx != MAX_TCP_OUT + 1)
   {
      tcp_out[lindx].int_call_typ = NOT_CONNECTED ;
   }

}
```

```
PCB     closepcb = {OP_ABORT,PROTO_TCP,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
;
char    tcp_close_err[] = "Error closing TCP link" ;

void    HANGUP_TCP_DEST(ushort loclid)
{

/*  This routine hangs up the TCP link */

   int ii,fnd_row ;


/*  Find the out_call row for this call */

   fnd_row = MAX_TCP_OUT + 1 ;
   for(ii = 0; ii < MAX_TCP_OUT && fnd_row == MAX_TCP_OUT + 1 ; ii++)
   {
      if (tcp_out[ii].int_call_typ != NOT_CONNECTED &&
           tcp_out[ii].cur_typ_id == loclid)
      {
         fnd_row = ii ;
      }
   }

/*  Close the apecified connection */

   closepcb.p_id = loclid ;
   closepcb.p_pid = fnd_row ;

   if (gattrace == 1 || gattrace == 3)
   {
      sprintf(obuf,"Closing TCP Link # %d, pid: %d",loclid,fnd_row) ;
      output_message(obuf) ;
   }


   if (TCPCTL(&closepcb) != 0)
   {
      output_message(tcp_close_err) ;
   }
}
```

```
void    HANGUP_ISDN_DEST(ushort loc_lid)
{
/*  This routine hangs up the isdn link given */

   int ii,fnd_row ;

/*  Disconnect the current link */

   if (gattrace == 1 || gattrace == 3)
   {
      sprintf(obuf,"ISDN hangup called on lid # %d",loc_lid) ;
      output_message(obuf) ;
   }

   DISC_ISDN_LINK(loc_lid) ;

/*  Find the current row in the isdn out_call structure */

   for (ii=0,fnd_row = -1;ii < MAX_ISDN_OUT && fnd_row == -1; ii++)
   {
      if (isdn_out[ii].int_call_typ != NOT_CONNECTED &&
         isdn_out[ii].cur_typ_id == loc_lid)
      {
         fnd_row = ii ;
      }
   }

/*  if an entry was found */
/*  Then */

   if (fnd_row != -1)
   {
/*       Clear the call from the internal out_call structures */

      isdn_out[fnd_row].int_call_typ = NOT_CONNECTED ;
   }
}
```

```
void    PROC_TCP_INCALL(ushort lindx)
{
/*  This routine processes a call from the TCP */

   char    tcp_f_buf[18] ;
   int     ii,fnd_row,cur_stat ;
   ushort  success,faultcode ;
   BCB     *curbcb ;

/*  If the status returned indicates a good listen */
/*  Then */
/*      Indicate TCP call received */

   success = 0 ;

   if (trans_tbl[lindx].tcpptr->p_stat == STAT_OK)
   {
      tcp_call_count++ ;

/*      Unpack the source address */

      CONV_TCP_RX(trans_tbl[lindx].tcpptr->p_fhost,tcp_f_buf) ;

/*      Choose the out_call row to use */

      fnd_row = -1 ;
      for (ii=0; ii < MAX_TCP_OUT && fnd_row == -1 ; ii++)
      {
         if (tcp_out[ii].int_call_typ == NOT_CONNECTED)
         {
            fnd_row = ii ;
         }
      }

/*      If out_call row was found */
/*      Then */
/*          Place the current connect id & entry in the row */

      if (fnd_row != -1)
      {
         tcp_out[fnd_row].cur_typ_id = trans_tbl[lindx].tcpptr->p_id ;
         tcp_out[fnd_row].tar_typ_entry = lindx ;

/*          Determine the call destination */

         success = DET_TCP_DEST(tcp_f_buf,&tcp_out[fnd_row],
                        trans_tbl[lindx].tcpptr->p_fport) ;

/*          if call destination check was succesful */
/*          then */
```

```
/*              Initiate call to destination */

        if (success != 0)

        {
           success = INIT_CALL_DEST(&tcp_out[fnd_row],TCPTYP) ;

/*              if call was initiated successfully */
/*              then */
/*                  Setup a receive for incoming data */

           if (success != 0)
           {
              curbcb = (BCB *) &globucmd[fnd_row] ;
              curbcb->b_id = trans_tbl[lindx].tcpptr->p_id ;
              curbcb->b_cnt = 128 ;
              curbcb->b_pid = fnd_row ;
              TCPRX(curbcb) ;

/*                      Setup the transmit buffer for future use */

              curbcb = (BCB *) &globucmd[MAX_ISDN_OUT+fnd_row] ;
              curbcb->b_id = trans_tbl[lindx].tcpptr->p_id ;
              curbcb->b_pid = fnd_row ;

           }
           else
           {
              sprintf(obuf,"Unable to initiate target call") ;
           }
        }
        else
        {
           sprintf(obuf,"Duplicate call to same destination is
refused");
        }
     }
     else
     {
        sprintf(obuf,"Exceeded maximum outstanding TCP calls") ;
     }

/*       If the transaction was not successful */
/*       Then */
/*           Hang up the TCP caller */

     if (success == 0)
     {
        HANGUP_TCP_DEST(trans_tbl[lindx].tcpptr->p_id) ;
        output_message(obuf) ;
```

```
      }

      TCP_LISTEN(lindx) ;

   }
   else
   {
      sprintf(obuf,"TCP listen error %d on address %s",
                trans_tbl[lindx].tcpptr->p_stat,
                trans_tbl[lindx].tcpaddr) ;
      output_message(obuf) ;
   }
}
```

```
int curpos = 0 ;
void    QUE_ISDN_ACT()
{
/*  This procedure queues isdn actions as they are received from the
DPP */
   int ii;


/*  If there is an action pending */
/*  Then */
/*      Determine subaction based on the pending information */

   if (isdn_call_sw == 1)
   {
      if (gattrace == 1 || gattrace == 3)
      {
         sprintf(obuf,"Received ISDN command packet, cmd: %d\n",
                                        globdat.dm.data_cmd) ;
         output_message(obuf) ;
      }

      switch(globdat.dm.data_cmd)
      {


/*          Case received data */
/*              Queue the received data to be processed */

         case ISDN_DATA :
            QUE_ISDN_RECEIVE() ;
            break ;

/*          Case received display message */
/*              Display the message to the operator */

         case ISDN_DISP :
            output_message(globdat.display.dis_Message) ;
            break ;

/*          Case received control information */
/*              Queue received control information */

         case ISDN_CONT :
            QUE_ISDN_CONTROL() ;
            break ;

/*          Case received open response */
/*              Copy data to open response buffer */
/*              Indicate open response received */
```

```
        case ISDN_OPEN :
           memcpy(&r,&globdat.dm,sizeof(r)) ;
           isdn_call_connected = 1 ;
           break ;
      }

/*       If gateway has not been shut down */
/*       Then */

      isdn_call_sw = 0 ;
      if (gateway_up == 1)
      {

/*           Initiate another wait for information */

        isdn_call_wait() ;
      }
   }
}
```

```
void    QUE_ISDN_CONTROL()
{
/*  This routine processes received isdn control messages */

   int ii ;
   struct timeb timebuffer ;

/*  Determine action based on received control code */

   if (gattrace == 1 || gattrace == 3)
   {
      sprintf(obuf,"ISDN control submessage type %d received",
                              globdat.ioctl.ioctl_CODE) ;
      output_message(obuf) ;
   }

   switch(globdat.ioctl.ioctl_CODE)
   {

/*       Case link clear indication */
/*           Que the hangup from the remote link */

      case ISDN_CLEAR :
         pend_act[frptr].act = OP_CLOSE ;
         pend_act[frptr].typ = ISDNTYP ;
         pend_act[frptr].indx = *(short *) &globdat.ioctl.ioctl_Data[0]
;

         if (gattrace == 1 || gattrace == 3)
         {
            sprintf(obuf,"Detected ISDN Remote close on link # %d",
                        pend_act[frptr].indx) ;
            output_message(obuf) ;
         }

         if (++frptr == MAX_PEND_ACT) frptr = 0 ;
         break ;

/*       Case incoming call indication */
/*           Que the incoming call indication for handling */

      case ISDN_INCALL :
         pend_act[frptr].act = OP_LISTEN ;
         pend_act[frptr].typ = ISDNTYP ;
         pend_act[frptr].indx = ircvncb.ncb_retcode ;
         ftime(&timebuffer) ;
         pend_act[frptr].qtime =timebuffer.time ;
         pend_act[frptr].msec = timebuffer.millitm ;
         if (++frptr == MAX_PEND_ACT) frptr = 0 ;
```

```
/*          Copy the incoming data into the incall buffer */

        memcpy(&incall.ioctl,&globdat.ioctl,sizeof(globdat.ioctl)) ;
        break ;
/*       Case statisics recieved */
/*          Copy the statistics returned into the local statistics
buffer */
/*          Indicate that the statistics have been received */

     case ISDN_STATS :
        isdn_stats_retcode = ircvncb.ncb_retcode ;
        isdn_stats_rx = 1 ;
        memcpy(&su,&globdat.dm,90) ;
        break ;

/*       Case default */
/*          Display a message to the operator indicating the receipt
*/

     default :

        sprintf(obuf,"ISDN msg: Code %d,",globdat.ioctl.ioctl_CODE) ;

        for(ii=0; ii <globdat.ioctl.ioctl_Size; ii++)
        {
          sprintf(obuf,"%s%hx ",obuf,globdat.ioctl.ioctl_Data[ii]) ;
        }
        output_message(obuf) ;
        break ;
   }
}
```

```
void    CONV_TCP_RX(ulong   fhost, char * svbuf)
{
/*  This routine converts a received TCP address to string */

   char     *sb ;
   char     *sa ;
   char     fmt_buffer[20];
   int      ii ;

/*  Loop for all 4 subparts */

   sb = svbuf ;
   sa = (char *) &fhost ;
   *sb = '\0' ;

   for(ii=0 ; ii < 4 ; ii++)
   {

/*       Convert the current subpart to ascii */
/*       Insert in buffer followed by a period */

      if (*sa < 10)
      {
         sprintf(fmt_buffer,"00%hd%s",(ushort) *sa++,sb) ;
      }
      else
      {
         if (*sa < 100)
         {
            sprintf(fmt_buffer,"0%hd%s",(ushort) *sa++,sb) ;
         }
         else
         {
            sprintf(fmt_buffer,"%hd%s",(ushort) *sa++,sb) ;
         }
      }
      if (ii < 3)
      {
         sprintf(sb,"%c%s",'.',fmt_buffer) ;
      }
      else
      {
         sprintf(sb,"%s",fmt_buffer) ;
      }

   }
}
```

```
ushort  DET_TCP_DEST(char   *rx_addr,OUT_CALL *curout,ushort port)
{
/*  This routine determines the destination of a TCP call */

   int ii,fnd_row ;
   ushort success ;

/*  Loop for all of the translation table entries */

   success = 0 ;
   fnd_row = -1 ;

   for(ii=0 ; ii < MAX_TBL && fnd_row == -1 ; ii++)
   {

/*       If current entry contains a TCP address and it matches the */
/*           source address of the call and the ports match */
/*       Then */
/*           Indicate that the row is found */

      if ((strcmp(rx_addr,trans_tbl[ii].tcpaddr) == 0) &&
          (trans_tbl[ii].tcpport == port || port > 99))
      {
         fnd_row = ii ;
      }
   }


/*  If entry is found */
/*  Then */
/*       Search the existing row for any additional outstanding calls
*/

   if (fnd_row != -1)
   {
      for (ii=0 ; ii < MAX_TCP_OUT && fnd_row != -1 ; ii++)
      {
         if (tcp_out[ii].int_call_typ != NOT_CONNECTED &&
              curout->tar_typ_entry == tcp_out[ii].tar_typ_entry &&
              tcp_out[ii].cur_typ_entry == fnd_row)
         {
            fnd_row = -1 ;
         }
      }

/*       If existing entry not found */
/*       Then */

      if (fnd_row != -1)
      {
```

```
/*          If both the destination and source have x25 addresses */
/*          Then */
/*              Indicate that the call is of type X.25 */
/*          Else */
/*              Indicate that the call is of type TCP */

        if (trans_tbl[fnd_row].x25addr[0] != '\0' &&
             trans_tbl[curout->tar_typ_entry].x25addr[0] != '\0')
        {
           curout->tar_typ = X25TYP ;
        }
        else
        {
           curout->tar_typ = TCPTYP ;
        }
        curout->cur_typ_entry = fnd_row ;
        success = 1 ;
      }
   }
   return(success) ;
}
```

```
ushort  INIT_X25_CALL(OUT_CALL *curout, ushort source_typ)
{
/*  This routine initiates calls to the X.25 destination */

   int      ii,fnd_row ;
   ushort   success ;
   ncb_t    opncb ;
   char     c, *tmpptr ;
   uchar    for_len,loc_len ;

/*  Find an available out_call row for this call */

   success = 0 ;
   fnd_row = -1 ;
   for (ii=0 ; ii < MAX_ISDN_OUT && fnd_row == -1 ; ii++ )
   {
      if (x25_out[ii].int_call_typ == NOT_CONNECTED)
      {
         fnd_row = ii ;
      }
   }

/*  If the row was found */
/*  Then */
/*       Perform common NCB actions */

   if (fnd_row != -1)
   {
      clear_ncb(&opncb) ;
      opncb.ncb_command = 0x10 ;
      destp = opncb.ncb_callname ;
      srcp = X25_NAME ;
      for (ii = 0 ; ii < 16 ; ii++, srcp++, destp++)
      {
         *destp = *srcp ;
      }
      opncb.ncb_buffer = call_buffer ;
      opncb.ncb_lana_num = 0x0ff ;

/*       Place the foreign address into the buffer */

      tmpptr = call_buffer ;
      for_len = CONV_X25_TX(trans_tbl[curout->tar_typ_entry].x25addr,
                            ++tmpptr,0) ;

/*       Place the local address in the buffer */

      tmpptr += (int) floor((for_len)/2) ;
      loc_len = CONV_X25_TX(trans_tbl[curout->cur_typ_entry].x25addr,
                          tmpptr,for_len-(2*floor((for_len/2)))) ;
```

```
/*      Insert the size information in the outgoing buffer */

     call_buffer[0] = (for_len << 4) + loc_len ;
     opncb.ncb_length = (for_len + loc_len + 1)/2 + 6 ;

     sprintf(obuf,"Call buffer contents:") ;
     for(ii=0; ii < (opncb.ncb_length-5) ; ii+=2)
     {
        sprintf(obuf,"%s %hx",obuf,*(short *)&call_buffer[ii]) ;
     }
     output_message(obuf) ;

/*      Insert facilities data */

     memcpy(&call_buffer[opncb.ncb_length - 5],perm_call_values, 5) ;

/*      Issue the command to the NIS */

     send_cmd_5c(&opncb) ;

/*      If the open was successful */
/*      Then */
/*          Setup the transmission buffer */

     if (opncb.ncb_retcode == 0)
     {

        if (gattrace == 1 || gattrace == 3)
        {
           sprintf(obuf,"Initiated X.25 call, link id %d",
                     opncb.ncb_lsn) ;
           output_message(obuf) ;
        }
        x25_out[fnd_row].cur_typ_entry = curout->tar_typ_entry ;
        x25_out[fnd_row].int_call_typ = source_typ ;
        x25_out[fnd_row].cur_typ_id = opncb.ncb_lsn ;
        x25_out[fnd_row].tar_typ = source_typ ;
        x25_out[fnd_row].tar_typ_entry = curout->cur_typ_entry ;
        x25_out[fnd_row].tar_typ_id = curout->cur_typ_id ;
        curout->tar_typ_id = opncb.ncb_lsn ;

        x25_call_setup(fnd_row) ;

/*          Setup Data Wait */

        x25_call_wait(fnd_row) ;

/*          Indicate call success */
```

```
            success = 1 ;

         }

/*       Else */
/*           Indicate call attempt was unsuccessful */

         else
         {
            sprintf(obuf,"X25 call open failure for %s",
                     trans_tbl[curout->tar_typ_entry].x25addr) ;
            output_message(obuf) ;
         }
      }
/*  Else */
/*       Indicate too many calls outstanding */

      else
      {
         output_message("Maximum number of outstanding X.25 calls
exceeded") ;
      }
      return(success) ;
}
```

```
ushort  CONV_X25_TX(char *addrptr, char *bufptr , ushort uplow)
{
/*  This routine converts an X.25 address to transmittable format */

   int     ii ;
   ushort  leng ;
   char    *lp,*ap ;
   uchar   tmpint ;

/*  Loop for the length of the string 2 at a time */
/*      Convert the current digit */
/*      Position in upper half of the buffer */


   ap = addrptr ;
   lp = bufptr ;
   leng = strlen(addrptr) ;

   for (ii=0 ; ii < (leng+uplow) ; ii += 2)
   {
      if (ii != 0 || uplow == 0)
      {
         tmpint = *ap++ - 48 ;
         tmpint = tmpint << 4 ;
      }
      else
      {
         tmpint = *lp ;
      }

/*       If last character not processed */
/*       Then */
/*           Insert the 2nd character in the lower nibble */

      if (ii != (leng+uplow)-1)
      {
         tmpint += *ap++ - 48 ;
      }
      *lp++ = tmpint ;
   }
   return(leng) ;
}
```

```
void    PROC_RX_TCP_DAT(ushort   lindx)
{
/*  This routine processes the reveived TCP data */

   ushort success ;


/*  If the data was received without error */
/*  Then */
/*      Update the TCP statistics accordingly */

   success = 0 ;
   if (globucmd[lindx].tcp_bcb.b_res > 0)
   {
      tcp_packets_recv++ ;
      tcp_bytes_recv+= globucmd[lindx].tcp_bcb.b_res ;

      memcpy(tcplpr,tcpbuf[lindx],globucmd[lindx].tcp_bcb.b_res) ;
      tcp_lpr_size = globucmd[lindx].tcp_bcb.b_res ;

      if (gattrace == 1 || gattrace == 3)
      {
         sprintf(obuf,"Received TCP data; link: %d Size %d",
                   tcp_out[lindx].cur_typ_id,tcp_lpr_size) ;
         output_message(obuf) ;
      }

/*       Transmit the data to the intended destination */

      success =  INIT_TX_DEST(&tcp_out[lindx],
                globucmd[lindx].tcp_bcb.b_res,tcpbuf[lindx],TCPTYP) ;

/*       if the transmission was successful */
/*       Then */
/*           Queue another receive */

      if (success != 0)
      {
         globucmd[lindx].tcp_bcb.b_cnt = 128 ;
         globucmd[lindx].tcp_bcb.b_flags = 0x0 ;
         TCPRX(&globucmd[lindx]) ;
      }
   }

/*  If some error has been encountered */
/*  Then */
/*       Hangup the destination link */

   if (success == 0)
   {
```

```
      output_message("Detected error processing TCP received data") ;
      HANGUP_DEST(&tcp_out[lindx]) ;
/*       Hangup TCP link */
      HANGUP_TCP_DEST(tcp_out[lindx].cur_typ_id) ;
   }
}
```

```
ushort     TX_X25_MSG(OUT_CALL *curout,ushort size, char far *bufptr,
                ushort source_typ)
{
/*  This routine transmits X.25 messages on request */

   int     ii,fnd_row ;
   ushort  success ;


/*  Find the out_call row for this call */

   success = 0 ;
   fnd_row = -1 ;
   for(ii=0 ; ii < MAX_X25_OUT && fnd_row == -1 ; ii++)
   {
      if (x25_out[ii].int_call_typ != NOT_CONNECTED &&
           x25_out[ii].cur_typ_id == curout->tar_typ_id)
      {
         fnd_row = ii ;
      }
   }

   if (gattrace == 1 || gattrace == 3)
   {
      sprintf(obuf,"Transmitting X.25 message,lid: %d, size %d",
             curout->tar_typ_id,size) ;
      output_message(obuf) ;
   }

/*  Copy the data to the output buffer */

   x25_call_setup(fnd_row) ;
   memcpy(ou[fnd_row].dm.data_Data,bufptr,size) ;
   oxncb[fnd_row].ncb_length = size ;

/*  Update the X.25 statistics */

   if (source_typ == TCPTYP)
   {
      brk_stats[X25TYP].rem1_bytes_recv += size ;
      brk_stats[X25TYP].rem1_packets_recv++ ;
   }
   else
   {
      brk_stats[X25TYP].rem2_bytes_recv += size ;
      brk_stats[X25TYP].rem2_packets_recv++ ;
   }

/*  Send the data on the X.25 */
```

```
   send_cmd_5c(&oxncb[fnd_row]) ;

/*  If the send was successful */
/*  Then */
/*      Update the link level statistics */

   if (oxncb[fnd_row].ncb_retcode == 0x00)
   {
      ovr_stats.rem1_packets_sent++ ;
      ovr_stats.rem1_bytes_sent += size ;
      success = 1 ;
   }
   else
   {
      sprintf(obuf,"X.25 link failure %d on LSN # %d",
                oxncb[fnd_row].ncb_retcode,curout->tar_typ_id) ;
      output_message(obuf) ;
   }
   return(success) ;
}
```

```
void    DET_ISDN_ACT(usnort qptr)
{
/*  This routine determines the actions required as a result of a
received */
/*      message from the DPP */

   struct timeb timebuffer ;


/*  Determine actions based on queued message */

   ftime(&timebuffer) ;

   switch(pend_act[qptr].act)
   {

/*      Case Recieved Isdn call */
/*          Process received Incoming call indication */

     case OP_LISTEN :
       PROC_ISDN_INCALL(pend_act[qptr].indx) ;
       break ;

/*      Case received data on the isdn link */
/*          If the data is an isdn call information header */
/*          Then */
/*              Process the info header to initiate the call */
/*          Else */
/*              Process the incoming data */

     case    OP_RX :
       tot_packs++ ;
       pack_queued_time += ((timebuffer.time -
pend_act[qptr].qtime)*1000)
                    + (((int) timebuffer.millitm) -
pend_act[qptr].msec) ;
       pend_act[qptr].qtime = timebuffer.time ;
       pend_act[qptr].msec = timebuffer.millitm ;
       if (isdn_out[pend_act[qptr].indx].int_call_typ
                          == IN_PROCESS)
       {
         PROC_ISDN_HDR(pend_act[qptr].indx) ;
       }
       else
       {
         PROC_RX_ISDN_DAT(pend_act[qptr].indx) ;
       }
       ftime(&timebuffer) ;
       pack_process_time += ((timebuffer.time -
pend_act[qptr].qtime)*1000)
```

```
                    + (((int) timebuffer.millitm) -
pend_act[qptr].msec) ;

        break ;
/*      Case received remote close indication */
/*          Close the destination link */

     case OP_CLOSE :
        PROC_ISDN_REMCLS(pend_act[qptr].indx) ;
        break ;

   }
}
```

```
void    PROC_ISDN_INCALL(ushort lindx)
{
/*  This routine processes received ISDN calls */

   int ii,fnd_row ;
   ncb_t   *locncbp ;
   ncb_t   acc_ncb ;

/*  Find an available out_call row for this call */

   fnd_row = -1 ;
   for (ii=0 ;  ii < MAX_ISDN_OUT && fnd_row == -1 ; ii++)
   {
      if (isdn_out[ii].int_call_typ == NOT_CONNECTED)
      {
         fnd_row = ii ;
      }
   }

/*  If out_call row was found */
/*  Then */
/*      Indicate that a call is expected for this row */

   if (fnd_row != -1)
   {
      isdn_out[fnd_row].int_call_typ = IN_PROCESS ;
      isdn_out[fnd_row].cur_typ_id =
                  *(short *) &incall.ioctl.ioctl_Data[46] ;

      if (gattrace == 1 || gattrace == 3)
      {
         sprintf(obuf,"Received ISDN call on lid %d ",
                  *(short *) &incall.ioctl.ioctl_Data[46]);
         output_message(obuf) ;
      }

/*      Accept the call */

      PERF_COM_NCB_ACT(&acc_ncb) ;
      acc_ncb.ncb_command = 0x14 ;
      acc_ncb.ncb_buffer = (char *) &incall ;
      acc_ncb.ncb_length = sizeof(incall) ;
      incall.ioctl.ioctl_cmd = 6 ;
      incall.ioctl.ioctl_CODE = isdn_out[fnd_row].cur_typ_id ;

      send_cmd_60(&acc_ncb) ;

/*      Update the link level statistics */

      ovr_stats.rem2_call_count++ ;
```

```
   }
   else
   {
      output_message("Maximum number of allowed ISDN circuits
exceeded") ;

/*       Reject the incoming call */

      DISC_ISDN_LINK(*(short *)&incall.ioctl.ioctl_Data[46]) ;

   }
}
```

```
void    PROC_ISDN_HDR(ushort lindx)
{
/*  This routine processes the ISDN header information to establish
the */
/*      next link in the requested call */

   ushort  success ;
   char    *sourceptr ;
   int     ii,fnd_row ;

/*  Get the message type from the source pointer */

   success = 0 ;
   sourceptr = iu[MAX_X25_OUT + lindx].dm.data_Data ;
      sprintf(obuf,"hdr packet: %s",sourceptr) ;
      output_message(obuf) ;
   isdn_out[lindx].tar_typ = (*sourceptr++ == 'X') ;

/*  Search for the target address in the translation table */

      sprintf(obuf,"hdr packet: %s",sourceptr) ;
      output_message(obuf) ;
   isdn_out[lindx].tar_typ_entry = CONV_ISDN_RX(sourceptr,
                                   isdn_out[lindx].tar_typ) ;

/*  If entry was found */
/*  Then */

   if (isdn_out[lindx].tar_typ_entry != -1)
   {

/*      Search for the source address in the translation table */

      sourceptr += (X25_ADDR_LEN + 2) ;
      sprintf(obuf,"hdr packet: %s",sourceptr) ;
      output_message(obuf) ;
      isdn_out[lindx].cur_typ_entry = CONV_ISDN_RX(sourceptr,
                                      isdn_out[lindx].tar_typ) ;


/*      If entry was found */
/*      Then */

      if (isdn_out[lindx].cur_typ_entry != -1)
      {

/*          Determine if this was a duplicate call */

         fnd_row = -1 ;
         for(ii=0; ii<MAX_ISDN_OUT && fnd_row == -1 ; ii++)
```

```
        {
           if (lindx != ii &&
              isdn_out[ii].int_call_typ > NOT_CONNECTED &&
              isdn_out[ii].cur_typ_entry ==
                 isdn_out[lindx].cur_typ_entry &&
              isdn_out[ii].tar_typ_entry ==
                  isdn_out[lindx].tar_typ_entry)
           {
              fnd_row = ii ;
           }
        }

/*           If call not duplicated */
/*           Then */
/*               Update statistics */

        if (fnd_row == -1)
        {
           ovr_stats.rem2_packets_recv++ ;
           ovr_stats.rem2_bytes_recv +=
                 iu[MAX_X25_OUT+lindx].dm.data_Size ;

/*               Initiate the call to the user */

           success = INIT_CALL_DEST(&isdn_out[lindx],
                     ISDNTYP) ;
        }
        else
        {
           output_message("Duplicate ISDN call received") ;
        }
     }
  }

/*  If call initiation not successful */
/*  Then */
/*      Hangup the ISDN call */

  if (success == 0)
  {
    HANGUP_ISDN_DEST(isdn_out[lindx].cur_typ_id) ;
  }
}
```

```
void    QUE_ISDN_RECEIVE()
{
/*  This routine quesues received ISDN data receive messages for
processing */

   int ii,fnd_row ;
   struct timeb timebuffer ;

/*  If the receive was successful */
/*  Then */
/*      Find the out_call row for this buffer */

   if (ircvncb.ncb_retcode == 0x00 || ircvncb.ncb_retcode == 0x06)
   {
      fnd_row = -1 ;
      for(ii=0; ii < MAX_ISDN_OUT && fnd_row == -1 ; ii++)
      {
         if (isdn_out[ii].int_call_typ != NOT_CONNECTED &&
            isdn_out[ii].cur_typ_id == globdat.dm.data_LID)
         {
            fnd_row = ii ;
         }
      }

/*       If the row was found */
/*       Then */
/*           Fill out the action queue entry */

      if (fnd_row != -1)
      {
         pend_act[frptr].act = OP_RX ;
         pend_act[frptr].typ = ISDNTYP ;
         pend_act[frptr].indx = fnd_row ;
         ftime(&timebuffer) ;
         pend_act[frptr].qtime = timebuffer.time ;
         pend_act[frptr].msec = timebuffer.millitm ;
         if (++frptr == MAX_PEND_ACT) frptr = 0 ;

/*           Copy the current information */

         memcpy(&iu[MAX_X25_OUT+fnd_row].dm,&globdat.dm,
                                 sizeof(globdat.dm));
      }
   }
   else
   {
      sprintf(obuf,"ISDN receive error # %d",ircvncb.ncb_retcode) ;
      output_message(obuf) ;
   }
```

```
}
int     CONV_ISDN_RX(char *bufptr, ushort tar_typ)
{
/*  This routine converts an address from the ISDN header and looks it
*/
/*      up in the translation table */

   ushort  tarport ;
   int     ii,fnd_row ;
   char    *sp,tar_addr[X25_ADDR_LEN+2],portnum[3];

/*  Extract the address from the current buffer position */

   sp = tar_addr ;
   for (ii=0 ; ii < X25_ADDR_LEN ; ii++)
   {
      if (*bufptr != ' ')
      {
         *sp++ = *bufptr ;
      }
      bufptr++ ;
   }
   *sp = '\0' ;


/*  If the current address is a TCP address */
/*  Then */
/*      Extract the TCP port number */

   if (tar_typ == TCPTYP)
   {
      strncpy(portnum,bufptr,2) ;
      portnum[2] = '\0' ;
      tarport = atoi(portnum) ;
   }


/*  Find the translation table entry */

   fnd_row = -1 ;
   for (ii = 0 ; ii < MAX_TBL && fnd_row == -1 ; ii++)
   {
      if ((tar_typ == TCPTYP &&
          strcmp(tar_addr,trans_tbl[ii].tcpaddr) == 0 &&
          tarport == trans_tbl[ii].tcpport) ||
         (tar_typ == X25TYP &&
          strcmp(tar_addr,trans_tbl[ii].x25addr) == 0))
      {
         fnd_row = ii ;
      }
```

```
    }
    return(fnd_row) ;
}
```

```
void    PROC_ISDN_REMCLS(ushort loclid)
{
/*  This routine processes a remote close received on an ISDN link */

   int ii,fnd_row ;

/*  Find the out_call row associated with this link id */

   fnd_row = -1 ;
   for (ii=0 ; ii <  MAX_ISDN_OUT && fnd_row == -1 ; ii++)
   {
      if (isdn_out[ii].int_call_typ != NOT_CONNECTED &&
         isdn_out[ii].cur_typ_id == loclid)
      {
         fnd_row = ii ;
      }
   }

/*  If found */
/*  Then */
/*       Hangup Destination link */

   if (fnd_row != -1)
   {
      HANGUP_DEST(&isdn_out[fnd_row]) ;
      isdn_out[fnd_row].int_call_typ = NOT_CONNECTED ;
   }
}
```

```
void    PROC_RX_ISDN_DAT(ushort lindx)
{
/*  This routine processes recevied ISDN data */

   ushort success ;

/*  Update the statistics accordingly */

   ovr_stats.rem2_bytes_recv += iu[MAX_X25_OUT+lindx].dm.data_Size ;
   ovr_stats.rem2_packets_recv++ ;
   lpi.dm.data_Size = iu[MAX_X25_OUT+lindx].dm.data_Size ;

   if (gattrace == 1 || gattrace == 3)
   {
      sprintf(obuf,"Received ISDN data packet on link # %d size %d",
           isdn_out[lindx].cur_typ_id,lpi.dm.data_Size) ;
      output_message(obuf) ;
   }

/*  Transmit the data to the chosen target */
   memcpy(lpi.dm.data_Data,iu[MAX_X25_OUT+lindx].dm.data_Data,
            iu[MAX_X25_OUT+lindx].dm.data_Size) ;
   success = INIT_TX_DEST(&isdn_out[lindx],
             iu[MAX_X25_OUT+lindx].dm.data_Size,
            iu[MAX_X25_OUT+lindx].dm.data_Data,ISDNTYP) ;

/*  If transmission was not successful */
/*  Then */
/*       Hangup Destination Link */

   if (success == 0)
   {
      HANGUP_DEST(&isdn_out[lindx]) ;

/*       Hangup the isdn link */

      HANGUP_ISDN_DEST(isdn_out[lindx].cur_typ_id) ;
   }
}
```

4.0 FUTURE ENHANCEMENTS

The intent of the following discussion is to illustrate the possible changes/enhancements to the field readiness and usability of the multiprotocol gateway. These changes fall into two major groupings. The first grouping relates to changes required to allow the system to support the full protocol definition for X.25 and TCP/IP links. The second set of changes are those changes that increase the efficiency or usability of the gateway. The relative complexity (degree of difficulty in implementing the change) of the change is shown using a scale from 1 to 4. A relative complexity of "1" represents approximately a three to four day job and a relative complexity of "4" represents approximately a four week job.

4.1 CHANGES FOR PROTOCOL COMPLETENESS

The changes outlined in this section correct deficiencies in the gateway implementation's use of the existing protocols.

4.1.1 X.25 Protocol Deficiencies

The table shown in Exhibit 4-1 presents the changes required for the gateway to perform the complete range of functions offered by the X.25 network interface card (NIC). The table presents the function, the relative complexity of the change to the existing gateway, and the current handling of the function.

4.1.2 TCP/IP Protocol Deficiencies

The table shown in Exhibit 4-2 presents the changes required for the gateway to perform the complete range of functions offered by the TCP/IP NIC. The table presents the function, the relative complexity of the change to the existing gateway, and the handling of the function.

4.2 CHANGES FOR EFFICIENCY AND EASE OF USE

The changes outlined in this section describe enhancements that are not required to field the system but that would improve its usability or throughput once it is installed in the field.

4.2.1 Initial Gateway Configuration

Changing the parameters of the gateway requires that constants in the gateway and in card configuration files be changed by the operator through use of an editor. In some cases the code must be recompiled before these changes can come into effect. There are three different types of parameters that may require change to configure the gateway to any new environment.

EXHIBIT 4-1

X.25 Enhancements

X.25 Protocol Feature	Relative Change Complexity	Current Handling
X.25 Facility Data	2	Ignored in incoming calls and discarded
X.25 Permanent Virtual Circuits	4	Not allowed
X.25 D Bit (end-to-end acknowledgement)	4	Ignored in incoming packets, information discarded
X.25 Cause and diagnostic codes	2	Ignored in incoming Resets, Hangups and Rejects, information discarded
X.25 User Data	1	Ignored in incoming calls and discarded
X.25 Aborts	1	Not used, current implementation allows all outbound data to be transmitted even in error situations
X.25 M Bit	1	Ignored in incoming packet, gateway does not repacketize data, currently discarded
X.25 Expedited Data	3	Data processed as normal packet, expedited indicator lost
X.25 Reset Request	2	Treated as link error causing link hang up

EXHIBIT 4-2

TCP/IP Enhancements

TCP Protocol Feature	Relative Change Complexity	Comments and Current Handling
Multiple Address Handling	Requires Vendor intervention	No changes to current gateway handling
Push Flag	1	Flag is ignored
Urgent Flag	1	All data messages currently treated with urgent priority (Note: same handling possible as with X.25 Expedited Data, difficulty assumes X.25 change already implemented
Fragmentation Flag	2	Gateway does not fragment packets, changes need only pass flag through; flag is currently ignored
TCP Close Handling	3	Gateway should be changed to use the Close command on normal close; currently aborts all connections
TCP Unspecified Ports	3	When an end user sends a message from an unspecified port the gateway matches the first instance for the user address only

They are:

- NIC configuration parameters. These parameters describe the card level parameters of the NIC. Changing these parameters requires modification of the "*.BAT" files prior to gateway startup as well as running the "X25NET CONFIG" command.

- Network operating parameters. These parameters, in particular for TCP/IP, describe the operating environment of the current network. These parameters are contained in the "NET.*" files in the TCP directory.

- Gateway operating parameters. These parameters describe such things as maximum outstanding calls, table sizes, and message sizes. These parameters are defined as "C" compile time constant and changes to them require recompile of the gateway program.

To eliminate this complexity when installing the gateway in a new environment a separate gateway configuration program should be created. This program would accept operator input about the configuration in a screen oriented fashion. It would then generate the required ".BAT," "NET.*" and *.CON" files in order to allow the gateway to run in the current environment. In addition the gateway program itself should be changed to read in its operating parameters online and, through use of dynamic memory, dynamically size internal tables. The relative complexity of this change and new development is 5 based on the previously used scale.

4.2.2 Subdivision of Gateway Functions

Network actions are performed as indivisible actions by the gateway driver. This is inefficient in that the time spent waiting for such actions as "Data Transmits" to complete could be used to process other incoming data. This could be improved by dividing these actions into smaller tasks with task boundaries at each of the NETBIOS calls. The gateway driver would have to be changed to be intelligent enough to determine which action the current NETBIOS response is part of and restart the action at its current point. This change would also require a change to use dynamically allocated queues and tables. Assuming the changes in section 4.2.1 were accomplished, the relative complexity of the this change is 4. The relative complexity of the overall change is 5 if the dynamic buffering is not in place.

4.2.3 Table Access Enhancements

The method for accessing the translation table and the outstanding call lists depends heavily on sequential searches. Use of this search method results in a heavy speed penalty as table size and load increases. The following changes are recommended:

- The source and destination outstanding call tables for a call should be linked by two way traversable pointers. Use of these pointers would eliminate the need to search the destination call table for each data packet received. The relative complexity of this change is 1.

- The search of the source table when data is received should be changed to be non-sequential. The relative complexity of this change is 1.

- The searches of the translation table should be changed to be more efficiently accomplished. Changes in the structure of the translation table should be designed to provide enhanced search efficiency. The relative complexity of this change is 2.

5.0 GATEWAY USERS GUIDE

5.1 INTRODUCTION

The gateway executable files are approximately 91,000 bytes plus approximately 520,735 bytes of executable files for the NICs. The recommended minimum hardware configuration for the multiprotocol gateway is a 286 or 386 based AT machine running DOS with the following components installed:

- 1 MB of onboard memory
- 1.2 MB floppy disk drive or hard disk
- 1 EICON X.25 PC card
- 1 Frontier Technologies Corporation AdCom2-I Intelligent Communications Controller
- 1 TELEOS B100PC Communications Coprocessor.

The gateway can also be implemented on a 286 or 386 machine with a 5.25 inch floppy disk drive. However, minor modifications are required to the ".BAT" files in the \EXEC directory as indicated in Appendix A to boot the gateway from multiple diskettes.

The following sections describe the various commands available to the gateway operator. These commands are organized into menus that are displayed during gateway operations. The commands available perform one of two main functions: (1) prepare the gateway for operation and (2) display the information collected during this operation.

5.2 INSTALLING THE GATEWAY

This section describes the gateway diskette that contains all gateway related software (refer to Appendix A for a listing of files) and contains information concerning initialization of the multiprotocol gateway.

5.2.1 Gateway Diskette

The gateway diskette contains the files required to change or execute the multiprotocol gateway. The file "readme.txt", located in the main directory of the diskette lists all of the files on the diskette. Execution of the gateway requires that the files listed in the "EXEC," "TCP," "X25" and "ISDN" directories be located on the disk or diskette and be accessible to the gateway program. The directory structure of the diskette can be changed from the one that exists on the diskette as long as the following steps are adhered to:

- Copy the files located in the EXEC directory to the target disk or diskette

- Copy the TCP protocol files, located in the TCP directory, to the desired directory on the target disk

- Modify the files "TCP_START.BAT" and "TCP_TRACE.BAT" to change their execution so that they move to the new TCP directory prior to execution and return to the new executable directory after execution

- Copy the X.25 protocol files to the desired directory on the target disk or diskette

- Modify the files "X25_START.BAT" and "X25_TRACE.BAT" as indicated above for the X.25 directory

- Copy the ISDN protocol files to the new ISDN directory

- Modify the files "ISDN_START.BAT" and "ISDN_TRACE.BAT" as indicated above for the ISDN directory.

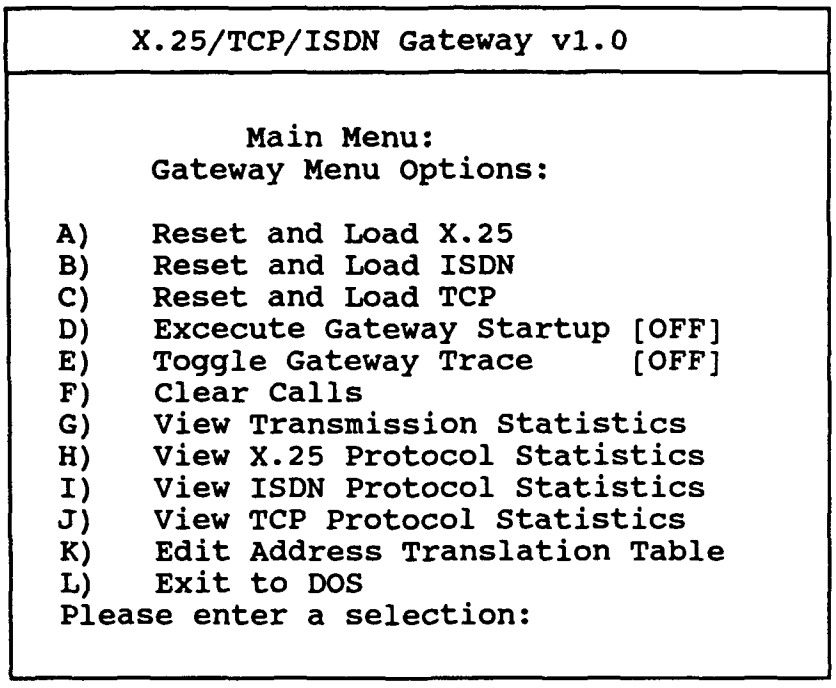
NOTE: File directories can be changed and/or combined but all the files that are grouped together in directories on the gateway diskette must stay in the same directory on the new disk or diskette.

Regardless of the directory structure the gateway must be modified for any changes in the NIC card configuration in a new host. If the NIC parameters have not changed then no configuration changes are required. If the Eicon X.25 NIC parameters have changed the user must change the NETBIOS program by entering the "X25NET CONFIG" command from the X.25 directory. Each of the changed parameters must be entered into the entry screen fields of the screen that is presented. An exit from this program will result in the update of the X.25 NETBIOS software. For ISDN and TCP NIC parameter changes the command line parameters appearing in the ".BAT" files for these protocols must be updated to indicate the new value. Reconfiguration of the driver software will not occur until the next time that card is loaded.

Once the above processes have been completed the multiprotocol gateway can be run by entering the command "GATEDEV" at the DOS command prompt. This command must be entered from the executable directory that contains the ".BAT" files. Proper start up of the gateway will display the main menu illustrated in Exhibit 5-1.

EXHIBIT 5-1

Main Menu Format



5.2.2 Gateway Use

Once the gateway has been installed and configured, the current set of end user addresses must be entered into the gateway translation table. Without these addresses the gateway operation will not start. Gateway translation table editing is discussed below. Once the addresses have been entered the operator should exit to DOS. This will cause the entered table values to be saved to a disk file for later use or for distribution to remote gateways. Each subsequent entry into the gateway program will use the stored values as long as the file is located in the gateway directory. This precludes the need to re-enter the values for each session. The gateway is now ready to be brought into operation as described below.

5.3 MAIN MENU

This section describes the Main Menu and the command options provided to the gateway operator.

5.3.1 Introduction

The Main Menu is the first menu presented to the gateway operator upon entry of the "Gatedev" command. This menu contains the complete list of command options available to the operator. Exhibit 5-1 illustrates the format of the Main Menu. Each command is initiated by entering the letter corresponding to the desired option. The following sections will discuss each of these commands in detail.

5.3.2 Card Load Commands

Options "A," "B," and "C" are the NIC card load commands. Option "A" resets the X.25 NIC and loads the NIC resident software. Option "B" resets the ISDN NIC and loads the NIC resident software. Option "C" resets the TCP/IP NIC and loads the NIC resident software. These commands require that there be at least one valid translation table entry prior to command execution. Each NIC that is required for use in the current gateway session must be loaded prior to issuance of the Gateway startup command. If a protocol is not required for the current session its NIC should not be loaded.

The completion of the NIC load command is signalled by a load completed message displayed on the Main Menu. If the status from that message is "0" then the command has completed successfully. If the status is not zero then some failure has occurred. To determine the failure, use the "E" option to clear the NIC tables, set the gateway trace to "[Control]" using the "F" key, and attempt to load the NIC again. This will display

the intermediate status and diagnostic messages on the gateway screen. When using the gateway software for the first time on a new host the card load commands should always be performed with the Gateway Trace function in the "[Control]" state. This will allow rapid determination of the correctness of the current NIC parameter settings.

5.3.3 Gateway Startup

Once all of the required cards have been loaded successfully the operator should use command option "D" to start up the gateway driver software. Once this command has been entered the gateway driver will register to receive incoming messages from the X.25 and TCP/IP links. The links registered are based on the Gateway Translation Table. Completion of the gateway startup will be signalled by the change of the gateway status indicator to "[ON]" from "[OFF]". Once this command has completed the gateway is fully operational.

5.3.4 Gateway Trace

Command option "E" toggles the trace facilities in the multiprotocol gateway. The Gateway Trace allows the operator to see on the screen, and log in a log file, the operational flow of the gateway as it processes incoming calls and data packets. The Gateway Trace facility has 4 states:

- "[OFF]" - gateway will only display and log command completions and error messages
- "[CONTROL]" - gateway will display and log control information as it processes the incoming calls and data along with the information indicated above
- "[DATA]" - gateway will log all data values passed through the gateway along with the command completion information and error messages described above
- "[BOTH]" - gateway will display and log all of the above information.

Gateway Trace state is set by pressing the "E" option key until the desired state is displayed in the status field following the command option. Trace information is displayed at the bottom of the Main Menu. Log information is stored in the file "Gatlog.log." This file is reset each time the gateway is executed. Use of the Gateway Trace function will significantly reduce gateway throughput and slow down all message traffic passing through it.

5.3.5 Gateway Shutdown

Shutdown of the gateway requires that the operator enter two separate Main Menu options. Option "F" should be entered to clear the calls that are outstanding. All calls will be cleared once the output activity on the call has been completed. Completion of this option is indicated by the gateway status indicator changing to "[OFF]." Option "L" saves the current translation table to disk and then exits to the DOS command level.

5.3.6 Gateway Statistics

There are two types of statistics available for display to the operator. These are available for display at any time during the gateway's operations. The two types are: (1) internal statistics kept by the gateway and (2) external statistics requested from the NIC.

The internal statistics kept by the gateway are:

- Number of calls processed
- Number of packets and bytes received
- Number of packets and bytes transmitted
- Last packet contents of each of the links.

These statistics are available in two different formats: overall and relative to a single protocol. Option "G" displays the overall internal statistics as well as the last packet contents for each of the links. Exhibits 5-2 and 5-3 show the format of these screens. For each of the protocol columns in Exhibit 5-2, command options "H" through "J" in the Main Menu provide further breakdown by source and destination protocols. Exhibits 5-4 through 5-6 show the format of these screens.

External statistics are available if there are open calls for the protocol selected under options "H" through "J". For each such call the gateway requests the externally held statistics package from the NIC and formats it for display.

5.3.7 Changing the Translation Table

Option "K" on the Main Menu allows the gateway operator to modify, delete or add entries in the gateway translation table. The translation table is used by the gateway as a basis for all translation activities. Each table entry consists of an X.25 address, a TCP address, a TCP port, and an ISDN address. The first three fields are optional depending on the protocol type of the described node. All entries require an ISDN address. Where one is not specified the gateway will assume that the node is on

EXHIBIT 5-2

Gateway Statistics Screen

```
                    X.25/TCP/ISDN Gateway v1.0

                    GATEWAY STATISTICS SCREEN


   Remote 1 [X.25]                                  Remote 2 [ISDN]

   Calls Proc'd : 0                                 Calls Proc'd : 0
   Packets Recv : 0                                 Packets Recv : 0
   Bytes Recv   : 0                                 Byte Recv    : 0
   Packets Sent : 0                                 Packets Sent : 0
   Bytes Sent   : 0                                 Bytes Sent   : 0
                          Remote 3 [TCP]

                          Calls Proc'd : 0
                          Packets Recv : 0
                          Bytes Recv   : 0
                          Packets Sent : 0
                          Byte Sent    : 0

Hit any key to proceed.
```

EXHIBIT 5-3

Gateway Statistics Screen
Last Packet Recieved

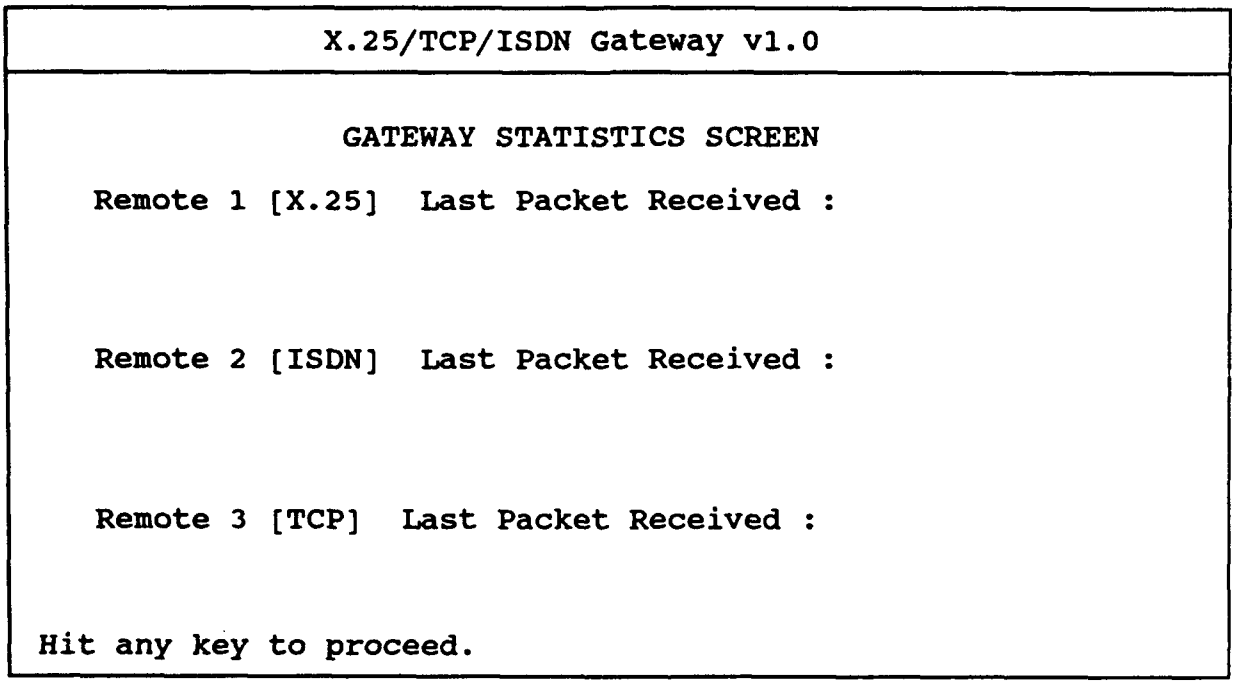


EXHIBIT 5-4

X.25 NIC Statistics

X.25/TCP/ISDN Gateway v1.0

X.25 STATISTICS SCREEN

Remote 1 [TCP]		Remote 2 [ISDN]	
Calls Proc'd	: 0	Calls Proc'd	: 0
Packets Recv	: 0	Packets Recv	: 0
Bytes Recv	: 0	Byte Recv	: 0
Packets Sent	: 0	Packets Sent	: 0
Bytes Sent	: 0	Bytes Sent	: 0

Hit any key to proceed.

EXHIBIT 5-5

ISDN NIC Statistics

```
                    X.25/TCP/ISDN Gateway v1.0

                        ISDN STATISTICS SCREEN


   Remote 1 [X.25]                          Remote 2 [TCP]

   Calls Proc'd : 0                         Calls Proc'd : 0
   Packets Recv : 0                         Packets Recv : 0
   Bytes Recv   : 0                         Byte Recv    : 0
   Packets Sent : 0                         Packets Sent : 0
   Bytes Sent   : 0                         Bytes Sent   : 0

    Total # of calls processed : 0



Hit any key to proceed.
```

EXHIBIT 5-6

TCP NIC Statistics

```
                    X.25/TCP/ISDN Gateway v1.0

                        TCP STATISTICS SCREEN


   Remote 1 [X.25]                          Remote 2 [ISDN]

   Calls Proc'd : 0                         Calls Proc'd : 0
   Packets Recv : 0                         Packets Recv : 0
   Bytes Recv   : 0                         Byte Recv    : 0
   Packets Sent : 0                         Packets Sent : 0
   Bytes Sent   : 0                         Bytes Sent   : 0

    Total # of calls processed : 0



  Hit any key to proceed.
```

the local gateway. Changes made to the translation table will not take effect until the next gateway startup.

5.3.7.1 Edit Session Screen

The Edit Session screen is displayed after the operator has entered the "K" option on the Main Menu. The Edit Session menu allows the operator to perform the high level functions associated with changing the translation table. Exhibit 5-7 shows the format of the Edit Session screen. The Edit Session menu is composed of the following options:

- Option "A" - requests that the gateway find an empty row in the translation table and allow the operator to enter a new entry
- Option "B" - requests that the gateway allow modification to the currently displayed table entry
- Option "C"&"D" - request that the gateway find the specified address in the table and allow the operator to update it
- Option "E" - requests that the next sequential table entry be displayed in the Edit Session screen
- Option "F" - requests that the program return to the main menu.

5.3.7.2 Entry Edit Screen

Edit Session options "A" through "D" allow the operator to modify the specified entry. This is accomplished through the Entry Edit screen. The format of this screen is displayed in Exhibit 5-8. When the screen is displayed the program will cycle through the value fields one at a time accepting input for each of the fields. Each address or port value must be entered completely, in the indicated format. The use of line edit keys is not allowed. To correct a given entry the operator must re-enter the entire address. Pressing the Enter key at the start of a field will indicate that the field is to remain the same. Pressing the Enter key at the end of the field will indicate that the value in the field is to be stored in the corresponding table entry field. Entering the Cntrl-Backsp key combination at the start of a field will result in the deletion of the field value. A row will be deleted from the table when all of the fields have been blanked. Pressing the Escape key will allow the operator to exit from the Entry Edit screen. The operator must cycle through all of the entries in the Entry Edit screen before exiting.

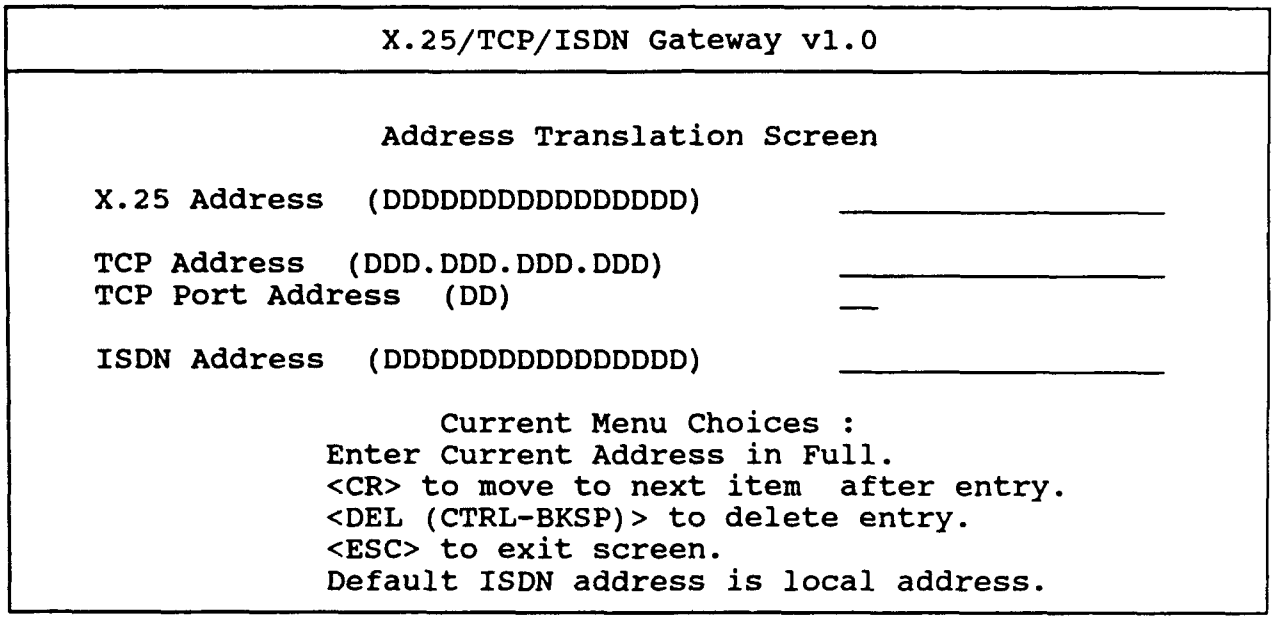
EXHIBIT 5-7

Edit Session Screen

```
                X.25/TCP/ISDN Gateway v1.0

                  Address Translation Screen

X.25 Address   (DDDDDDDDDDDDDDD)        8846440145_______

TCP Address    (DDD.DDD.DDD.DDD)        010.000.000.051__
TCP Port Address  (DD)                  1_

ISDN Address   (DDDDDDDDDDDDDDD)        0002_____________

                    Current Menu Choices :
                  A) Create New Entry.
                  B) Edit Current Address.
                  C) Edit Specified X.25 Address.
                  D) Edit specified TCP Address.
                  E) Scroll to Next Address.
                  F) Exit to Main Menu.
```

EXHIBIT 5-8

Entry Edit Screen



6.0 SUMMARY

The Multiprotocol Gateway Capabilities Demonstration Report and the Multiprotocol Gateway Optimization Report describe the hardware and software necessary to implement the multiprotocol gateway. Together with the vendor documentation associated with each NIC, these documents contain the necessary information to configure and operate the gateway. The gateway software is contained on a 1.2 MB gateway diskette (refer to Section 3 and Appendix A). The recommended minimum hardware configuration is a 286 or 386 based AT machine running DOS with the following components installed:

- 1 MB of onboard memory
- 1.2 MB floppy disk drive or hard disk
- 1 EICON X.25 PC card
- 1 Frontier Technologies Corporation AdCom2-I Intelligent Communications Controller
- 1 TELEOS B100PC Communications Coprocessor.

The gateway can be implemented on a 286 or 386 machine with a 5.25 inch floppy disk drive. However, minor modifications are required to the ".BAT" files in the \EXEC directory as indicated in Appendix A to boot the gateway from multiple diskettes.

The gateway program currently runs with full EGA graphics. However, it can also run with monochrome systems. The program can be run from a floppy disk as well as a hard disk. The gateway executable files are approximately 91,000 bytes plus approximately 520,735 bytes of executable files for the NICs. The gateway can also be used as an ISDN end user, an X.25 end user, or a DDN X.25 (TCP/IP) end user terminal providing that the appropriate vendor software is loaded.

It should be noted that the multiprotocol gateway software is the result of a proof-of-concept gateway development. There are several features that can be incorporated into the gateway to enhance the field readiness and usability of the multiprotocol gateway and to make the gateway more functional. These are described in Section 4. From a NCS perspective, these features should be researched and implemented because they provide insight and guidance on how these features should be provided for in a production version of the gateway.

A P P E N D I X A

```
************************************************************
*  The following file lists the files which are contained  *
*  in the delivered gateway diskette. This diskette        *
*  contains all of the necessary files to run or further   *
*  develop the multiprotocol gateway                       *
*      SWM 200390 Booz, Allen & Hamilton                   *
************************************************************
```

DIRECTORY PATH LISTING

Files:	COMMAND	.COM - DOS Command file
	README	.TXT - Contains this text

Path: \CODE

```
*************************************************************
*  This directory contains the "C" program files which make*
*  up the Multi-protocol Gateway. These files were created *
*  using MicroSoft Quick "C". The gateway is compiled as   *
*  4 seperate "C" modules as indicated below. All other    *
*  files are included with #include statements. The program*
*  is loaded using the "medium" memory model under the     *
*  MicroSoft "C" segmented memory mapping system           *
*************************************************************
```

Sub-directories: None

Files:

ISDN_STA.C	
GATEDEV .C	- Main Program Module- contains elements of the Man-Machine Interface module as well as the Data Transfer and Gateway Driver Modules. Compiled as a seperate "C" module.
STUBS .C	- Contains Stubs purposely left in the Gateway for further enhancement.
IN_TC_LI.C	
WA_US_IN.C	
ED_TR_TB.C	
DISPSCRN.C	
DE_ED_AC.C	
GE_IN_AD.C	
ED_SE_RW.C	
UP_X2_AD.C	
UP_TC_AD.C	
UP_IS_AD.C	
ST_TR_TB.C	
RD_TR_TB.C	
DI_BR_ST.C	
TCP_STAT.C	
SET_GATE.C	
ED&STATS.C	- Editing and Statistics Task file - Compiled as a seperate "C" module.
IN_TC_LS.C	
TCPLISN .C	
CO_TC_TX.C	
IN_X_LIS.C	
INITCOD .C	- Gate State Handler Module file - Compiled as a seperate "C" module.
CHK_NET .C	
DET_X_AC.C	
PR_X_INC.C	

```
DET_X_DS.C
IN_CA_DS.C
COM_IS_A.C
IN_IS_CA.C
DS_IS_LI.C
FM_IS_HD.C
Q_TC_ACT.C
IN_TC_CA.C
PRX_X_DA.C
IN_TX_DS.C
HANGDST .C
TX_IS_MS.C
CALLPROC.C  - Call Processing Module file -
                Compiled as a seperate "C" module.
TX_TC_MS.C
DET_T_AC.C
PTX_TC_C.C
PR_TC_CL.C
HNG_T_DS.C
HNG_I_DS.C
PR_T_CAL.C
Q_IS_ACT.C
Q_IS_CON.C
CON_T_RX.C
DET_T_DS.C
IN_X_CAL.C
CON_X_TX.C
PRX_T_DA.C
TX_X2_MS.C
DET_I_AC.C
PR_I_CAL.C
PR_I_HDR.C
Q_IS_RCV.C
CON_I_RX.C
PR_I_RCL.C
PRX_I_DA.C
```

Path: \CODE\FRONTER

```
************************************************************
*  The following directory contains the Frontier Software  *
*  Interface routines as well as associated constant files *
*  which are used in the Gateway Code                      *
************************************************************
```

Sub-directories: None

Files:

```
FTCTYPES.H
ULPUSER .H
ULPSHARE.H
NETERR  .C
TCPCTL  .C
ULPRXCMD.C
TCPRX   .C
TCPTX   .C
```

Path: \CONST

```
*************************************************************
*  This directory contains constant definition files used   *
*  in the Gateway Software.                                 *
*************************************************************
```

Sub-directories: None

Files: GATEDEF .H - Gateway Value Constants Definition
TIPEDEF .H - Gateway Type definitions
INCHDR .H - Common includes for C module files
GLOBALDE.H - External Global Variable Definitions
SUBDECL .H - External Subroutine Definitions

Path: \EXEC

```
************************************************************
*  This directory contains the basic files required to     *
*  execute the gateway. The command "gatedev" must be      *
*  entered from this directory. These files must be in the *
*  base directory wherever the gateway is loaded.          *
************************************************************
```

Sub-directories: None

Files: SETUP .BAT - Setup file for Quick "C" compiler environment

GATEDEV .MAK - Make file for the Gatedev system

GATEDEV .EXE - Gateway executable file

X25_STAR.BAT - X.25 Startup batch file (No Trace). Assumes Directory Structure as defined on this diskette. Must be modified when system is ported.

X25_TRAC.BAT - X.25 Startup batch file (Trace). Assumes Directory Structure as defined on this diskette. Must be modified when system is moved.

ISDN_STA.BAT - ISDN Startup batch file (No Trace). Assumes Directory Structure as defined on this diskette. Must be modified when system is moved. Contains Card Parameter values which may required change.

ISDN_TRA.BAT - ISDN Startup batch file (Trace). Assumes Directory Structure as defined on this diskette. Must be modified when system is moved. Contains Card Parameter values which may require change.

TCP_TRAC.BAT - TCP Startup batch file (Trace). Assumes Directory Structure as defined on this diskette. Must be modified when system is moved. Contains Card Parameter values which may require change.

TCP_STAR.BAT - TCP Startup batch file (No Trace).

Assumes Directory Structure as defined
on this diskette. Must be modified when

system is moved.Contains Card Parameter

values which may require change.

GATTRANS.TBL - Example translation storage format. Please

delete before actual use.

GATLOG .LOG - Example trace output file.

Path: \BIN\X25

```
*****************************************************************
*  This directory contains the executables and binarys          *
*  required to bring the EICON X.25 card online. These          *
*  programs are activated by the .BAT files in the \EXEC        *
*  path under Gateway control. The .BAT files for X.25 must     *
*  be changed to point to the directory containing these        *
*  files if the system is installed in a different              *
*  directory structure                                          *
*****************************************************************
```

Sub-directories: None

Files:

FORMIX	.COM	
NABIOS	.COM	
X25NET	.EXE	- This command is also used to configure the EICON card for the current card parameter settings. Enter the command from the command line as "X25NET CONFIG"
X25NET	.CFG	
X25NET	.IMG	
X25CFG	.FMX	
NATEST	.EXE	
NATEST	.IMG	
ACCCFG	.EXE	

Path: \BIN\TCP

```
*****************************************************************
*  This directory contains the executables and binarys          *
*  required to bring the Adcom2i TCP card online. These         *
*  programs are activated by the .BAT files in the \EXEC        *
*  path under Gateway control. The .BAT files for TCP must      *
*  be changed to point to the directory containing these        *
*  files if the system is installed in a different              *
*  directory structure                                          *
*****************************************************************
```

Sub-directories: None

Files:

FTX	.BIN
L2	.BIN
L3M1	.BIN
PROCESS	.BIN
TCP	.BIN
LOADF	.EXE
INST[illegible]CP	.EXE
[illegible]T2I	.EXE
[illegible]TUP	.EXE
NET	.X25
NET	.TCP
TCP	.ORD

Path: \BIN\ISDN

```
*****************************************************************
*  This directory contains the executables and binarys          *
*  required to bring the Teleos ISDN card online. These         *
*  programs are activated by the .BAT files in the \EXEC        *
*  path under Gateway control. The .BAT files for ISDN must     *
*  be changed to point to the directory containing these        *
*  files if the system is installed in a different              *
*  directory structure                                          *
*****************************************************************
```

Sub-directories: None

Files:

```
BCONFIG .EXE
BSTART  .EXE
CONFIG  .SIG
CONFIG  .VFI
NETBIOS .EXE
SIGNAL  .EXE
```

Path: \FORMATD

```
*****************************************************************
*  The following files are WordPerfect formatted code files     *
*  formatted to allow the files to be printed properly.         *
*  These files should be used only to provide hardcopy          *
*  versions of the code, not as a basis for development         *
*****************************************************************
```

Sub-directories: None

Files:

```
CON_T_RX.C
COM_IS_A.C
DET_T_AC.C
CO_TC_TX.C
DET_X_AC.C
DE_ED_AC.C
DI_BR_ST.C
INITCOD .C
ED_TR_TB.C
HANGDST .C
HNG_T_DS.C
PR_X_INC.C
IN_TC_CA.C
IN_X_CAL.C
```

IN_TC_LS.C
ISDN_STA.C
PRX_T_DA.C
PRX_X_DA.C
PR_I_HDR.C
PR_TC_CL.C
GATEDEV .C
Q_IS_ACT.C
Q_IS_RCV.C
RD_TR_TB.C
TCPLISN .C
TX_IS_MS.C
TX_X2_MS.C
UP_TC_AD.C
WA_US_IN.C
ED&STATS.C
CHK_NET .C
CON_I_RX.C
CON_X_TX.C
DET_I_AC.C
DET_T_DS.C
DET_X_DS.C
DS_IS_LI.C
ED_SE_RW.C
FM_IS_HD.C
GE_IN_AD.C
HNG_I_DS.C
IN_CA_DS.C
IN_IS_CA.C
IN_TC_LI.C
IN_TX_DS.C
IN_X_LIS.C
PRX_I_DA.C
PR_I_CAL.C
PR_I_RCL.C
PR_T_CAL.C
PTX_TC_C.C
Q_IS_CON.C
Q_TC_ACT.C
SET_GATE.C
ST_TR_TB.C
TCP_STAT.C
TX_TC_MS.C
UP_IS_AD.C
UP_X2_AD.C
DISPSCRN.C
CALLPROC.C

LIST OF ACRONYMS

L I S T O F A C R O N Y M S

ACRONYM	DESCRIPTION
BRI	Basic Rate Interface
bps	Bits per second
CCITT	The International Telegraph and Telephone Consultative Committee
DCE	Data circuit-terminating equipment
DPP	Data Protocol Processing
DTE	Data terminal equipment
DDN	Defense Data Network
DOD	Department of Defense
FIFO	First In First Out
GTM	Gateway Translation Matrix
ID	Identifier
ISDN	Integrated Services Digital Network
IP	Internet Protocol
kbps	Kilobits per second
kB	Kilobytes
MB	Megabytes
MPU	Master Processing Unit
MMI	Man-Machine Interface
NS/EP	National Security and Emergency Preparedness
NIC	Network Interface Card

OMNCS	Office Of The Manager, National Communications System
PC	Personal Computer
RAM	Random Access Memory
SDL	Specification and Description Language
SIP	Signal Input Processor
TA	Terminal Adaptor
TCP	Transmission Control Protocol